THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR [illegible]

TITLE OF THESIS [illegible]

DEGREE FOR WHICH THESIS WAS PRESENTED [illegible]

YEAR THIS DEGREE GRANTED [illegible]

Permission [illegible] OF ALBERTA [illegible] this thesis [illegible] private [illegible] only.

[illegible]

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR Mary M. Higginson

TITLE OF THESIS Exact Methods for Systems of Polynomial Equations

DEGREE FOR WHICH THESIS WAS PRESENTED Master of Science

YEAR THIS DEGREE GRANTED 1974

THE UNIVERSITY OF ALBERTA

EXACT METHODS FOR SYSTEMS OF POLYNOMIAL EQUATIONS

by

© MARY MARGARET HIGGINSON

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

SPRING, 1974

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled "EXACT METHODS FOR SYSTEMS OF POLYNOMIAL EQUATIONS" submitted by MARY M. HIGGINSON in partial fulfilment for the degree of Master of Science.

ABSTRACT

The problem discussed is the exact computer solution of systems of linear equations whose coefficients are integers or polynomials over the integers. Two methods, the multi-step and congruential algorithms, are described and compared. The number of single-precision operations required to perform each is found, and it is concluded that the congruential method is superior except perhaps for small systems.

In previous congruential algorithms for the polynomials, certain 'bad' primes which occur are discarded. In this study it is shown that these primes need not be discarded. In addition, a theorem for effectively terminating the Chinese Remainder formula for polynomials is given.

ACKNOWLEDGMENTS

I wish to express my appreciation to my supervisors, Dr. S. Cabay and Dr. T.A. Marsland, for their encouragement and support, and for their criticism.

The financial assistance provided by the Department of Computing Science and by the National Research Council of Canada is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE

Introduction

The problem to be considered in this study is the exact computer solution of systems of linear equations with coefficients in the integers I , or in $I(x_1,\ldots,x_r)$, the multivariate polynomials over the integers.

Symbolic solutions are required in connection with a wide range of problems. The applications of flowgraphs serve as an illustration. A flowgraph is a labelled, directed graph with a starting node and a final node. The generating function G of a flowgraph F having starting node s , final node f , and transmission matrix T can be found by solving the system of n equations $Ax = b$ for x_n , where $A = (I-T^{\text{transpose}})$ and b is the s'th unit vector. Problems arising in electric circuit theory, in the analysis of Markov chains, in graph theory, and in coding theory can all be solved by finding such a generating function. The graph in Figure 1, for example, may be considered a flowgraph with starting node $s=1$, final node $f=6$, and transmission (or connection) matrix $T=(t_{ij})$; t_{ij} is the label on the path from node i to node j . Solution of the system of equations

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -a & 1 & -b & -a & -a & 0 \\ -b & -a & 1 & 0 & 0 & -b \\ 0 & -b & 0 & 1 & -b & 0 \\ 0 & 0 & -a & 0 & 1 & -a \\ 0 & 0 & 0 & -b & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

yields

$$G = x_6 = \frac{ab^2+a^3b^2+ab^3+a^2b^3+b^4}{-2ab-a^3b-ab^2-a^2b^2-ab^3-b^4-a^3+1} .$$

Two possible interpretations of this flowgraph, and of G, follow.

If the graph in Figure 1 is interpreted as a Markov graph, where t_{ij} = probability of a transition from state i to state j, then the k'th coefficient of the expansion of G represents the probability of a transition from starting state 1 to final state 6 in k steps. For example, if $a = \frac{3}{4} z$ and $b = 1 - \frac{3}{4} z = \frac{1}{4} z$, then

$$G = \frac{\frac{3}{64} z^3 + \frac{1}{64} z^4 + \frac{9}{256} z^5}{-\frac{3}{8} z^2 - \frac{15}{32} z^3 - \frac{5}{32} z^4 + 1} .$$

The probability of reaching state 6 in, say, 5 steps is found by expanding G to obtain the coefficient of z^5, which is 0.05273437 .

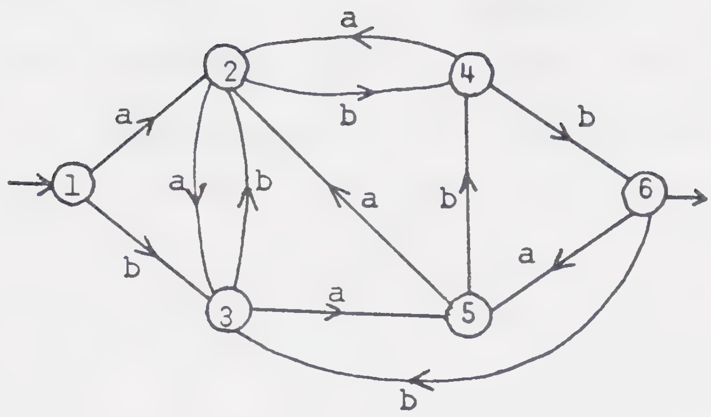


FIGURE 1. FLOWGRAPH 1

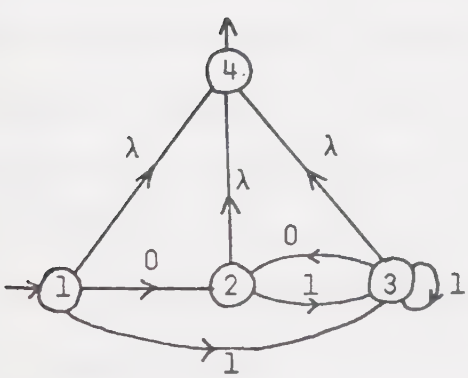


FIGURE 2(a). MACHINE

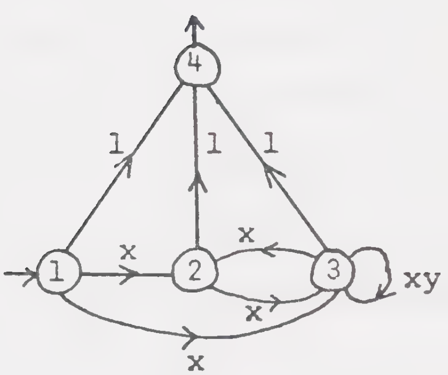


FIGURE 2(b). FLOWGRAPH 2

If, on the other hand, the graph in Figure 1 is an adjacency graph in which $a = b = z$, and $t_{ij} = z$ indicates that there is a path from node i to node j , then the k'th coefficient of G is the number of paths of length k from starting node 1 to final node 6 . In this case,

$$G = \frac{z^3 + 2z^4 + 2z^5}{-2z^2 - 2z^3 - 4z^4 + 1} = z^3 + 2z^4 + 4z^5 + \dots \quad .$$

There are no paths of length 0, 1, or 2 from node 1 to node 6 , 1 of length 3 , 2 of length 4 , and so on.

As another example, consider the problem of finding the number of n-digit binary sequences with exactly r pairs of adjacent 1's and no adjacent 0's . A finite state machine which recognizes sequences having no adjacent 0's is shown in Figure 2(a). In Figure 2(b), the corresponding flowgraph is shown; the branches are labelled with bivariate monomials $x^{e_1}y^{e_2}$, where $e_1 = 1$ indicates the occurrence of a 0 or 1 , $e_1 = 0$ indicates no occurrence of a 0 or 1 , $e_2 = 1$ indicates the occurrence of the sequence 11 , and $e_2 = 0$ indicates no occurrence of 11 . For this flowgraph,

$$G = \frac{1 - (y-2)x - (y-1)x^2}{1 - yx - x^2} = 1 + 2x + (y+2)x^2 + (2y+y^2+2)x^3 + \dots \quad .$$

The coefficient of $x^n y^r$ is the number of sequences of length n having exactly r pairs of adjacent 1's. There are, for example, 2 sequences of length 3 having exactly 1 pair of adjacent 1's and no adjacent 0's.

Flowgraphs are used in coding theory to enumerate comma-free code words, and in electric circuit theory to find the transmission of a circuit. The examples above, and others, are given by Lipson [20].

The traditional method for solving systems of linear equations is Gaussian elimination. In this algorithm, an element a_{ik} of the coefficient matrix is reduced to zero by subtracting from row i, $(a_{ik}/a_{kk}) \times$ pivot row k. This method cannot be used for integer or polynomial systems since division is not defined in an integral domain.

To overcome this problem, Rosser [25] has discussed an obvious modification of the Gaussian elimination method whereby one successively subtracts $a_{kk} \times$ row i from $a_{ik} \times$ row k. The method no longer requires division. However it is unsatisfactory because the precision of the integers (or the degree of the polynomials) doubles at each step. As a result of this exponential growth of coefficients, the method requires the equivalent of $O(n^3 \cdot 2^n)$ single-precision operations to solve a system of n linear equations with single-precision integer coefficients. Three alternatives to this method have been proposed.

The growth of coefficients can be minimized by finding the greatest common divisor of a_{kk} and a_{ik} at each step. Algorithms which find the g.c.d. of each column of the coefficient matrix (and perform row operations accordingly) have been investigated by Rosser [25], Blankinship [6,7] and Bradley [11,12]. Although the growth of coefficients is linear, the computation required to find the g.c.d.'s is large, and the complexity of the method is $O(n^5)$.

A compromise to removing all common factors is found in the multi-step algorithms. These algorithms are a modification of fraction-free elimination which remove at each step a factor which can be shown to occur. Although the largest factor is not always removed as in the g.c.d. algorithm, the rate of growth of the coefficients remains linear, and now no work is involved in finding the factor which is known before-hand. As a consequence, multi-step methods are superior to g.c.d. methods. Multi-step algorithms are discussed in Chapter Two and it is shown that the complexity is $O(n^5)$ or $O(n^4 \log n)$, depending on the multi-precision arithmetic algorithms used.

Another alternative exists: the congruential algorithms solve the given system modulo a number of single-precision primes and construct a solution from the modular solutions by means of the Chinese Remainder

Theorem. Several solutions have to be found, but they can be computed using single-precision operations only. Surprisingly, the requisite number of solutions can be found, and the solution constructed, in only $O(n^4)$ operations. This fact is shown in Chapter Three, and some improvements to the existing algorithms are suggested.

There has been some controversy over which of the latter two methods, multi-step or congruential, is actually superior. In this study, both methods are investigated with the objective of determining which should be the focus of future research. In Chapter Two the one and two-step algorithms are considered in detail; exact operation counts for the integer case are found, and the asymptotic behaviour of the polynomial case is established. The congruential algorithms are similarly treated in Chapter Three.

Finally, in Chapter Four it is shown that the congruential method is asymptotically superior to the multi-step methods, both for systems with integer coefficients and systems with polynomial coefficients. It is also established that for systems of n linear equations with single-precision integer coefficients, the number of operations required by the congruential methods will always be less than those required by the two-step algorithm if n is greater than about five. Similar results in the polynomial case are difficult to establish for

small n because the number of operations depends not only on n but also on the number of variables, the degrees of these variables, and the number of terms. Assumptions made in order to simplify the relationships among these factors, assumptions which typify real-world problems, tend to make results unreliable for small n . However, generalizing on the results obtained for the integer case, the congruential method should be superior to the multi-step methods for similar, and perhaps even smaller, values of n . Experimental results are necessary in establishing more definitive conclusions,and McClellan [23] is currently working on such experiments.

Since congruential algorithms are superior to multi-step algorithms for all but small n , special attention is given to these algorithms. One disadvantage of congruential algorithms is that certain modular solutions may have to be discarded if 'bad' primes occur. The primes referred to here are the moduli with respect to which the system is solved. In Chapter Three a major effort is made to deal with such primes. Also in Chapter Three a theorem for effectively terminating the Chinese Remainder formula for polynomials is given. The results obtained there are perhaps the highlights of this thesis.

CHAPTER TWO

The Multi-Step Methods

Discovery of the one-step fraction-free algorithm is attributed to Jordan (1838-1922). More recently, Bodewig [8], Fox [15], and Luther and Guseman [22] have discussed this method of solving exactly a system of linear equations with integer coefficients. The two-step algorithm was proposed by Bareiss [1,2,3,4] and has also been discussed for systems either with integer or with polynomial coefficients by Lipson [20].

In section 2.1 the algorithms are discussed. In section 2.2 the exact number of operations required to solve an integer system by the one and two-step algorithms is calculated, while in section 2.3 the approximate number of operations required for systems with polynomial coefficients is determined. The order of complexity of the multi-step algorithms is considered in section 2.4.

The multi-precision arithmetic required by these algorithms can be performed using either the 'classical' algorithms such as those described by Knuth [19, section 4.3.1], or various 'fast' techniques which have been recently developed. In sections 2.2 and 2.3, the analysis assumes the 'classical' algorithms are used; the order of

complexity which can be achieved using 'fast' algorithms is discussed in section 2.4.

2.1 The Algorithms

The one-step algorithm is a fraction-free Gaussian elimination algorithm which reduces the rate of growth of the coefficients by removing at each step a factor which is known to occur. To triangularize the n by $(n+1)$ augmented matrix $[A,b] = (a_{ij}^{(0)})$ of a system of n linear equations $Ax = b$, the algorithm is as follows:

$$a_{00}^{(-1)} = 1 \;;$$

For $k=1,2,\ldots,n-1$;

For $i=k+1,k+2,\ldots,n$;

For $j=k+1,k+2,\ldots,n+1$;

$$\text{Do } a_{ij}^{(k)} = \frac{a_{kk}^{(k-1)} a_{ij}^{(k-1)} - a_{kj}^{(k-1)} a_{ik}^{(k-1)}}{a_{k-1,k-1}^{(k-2)}} ,$$

where it is understood that

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)} & \text{for } 1 \le i \le k \;, \quad 1 \le j \le n+1 \\ 0 & \text{for } k+1 \le i \le n \;, \quad 1 \le j \le k \end{cases} .$$

The division is exact (i.e. fraction-free) for coefficients in any integral domain. (See Lipson [20]).

The two-step algorithm performs in one iteration two steps of the one-step algorithm. The pivot row k is calculated as in the one-step method, while for rows $(k+1)$ to n

$$a_{ij}^{(k)} = \left(a_{kk}^{(k-1)} a_{ij}^{(k-1)} - a_{kj}^{(k-1)} a_{ik}^{(k-1)}\right) / a_{k-1,k-1}^{(k-2)}$$

$$= \left[\frac{a_{k-1,k-1}^{(k-2)} a_{kk}^{(k-2)} - a_{k-1,k}^{(k-2)} a_{k,k-1}^{(k-2)}}{a_{k-2,k-2}^{(k-3)}} \cdot \frac{a_{k-1,k-1}^{(k-2)} a_{ij}^{(k-2)} - a_{k-1,j}^{(k-2)} a_{i,k-1}^{(k-2)}}{a_{k-2,k-2}^{(k-3)}} \right.$$

$$\left. - \frac{a_{k-1,k-1}^{(k-2)} a_{kj}^{(k-2)} - a_{k-1,j}^{(k-2)} a_{k,k-1}^{(k-2)}}{a_{k-2,k-2}^{(k-3)}} \cdot \frac{a_{k-1,k-1}^{(k-2)} a_{ik}^{(k-2)} - a_{k-1,k}^{(k-2)} a_{i,k-1}^{(k-2)}}{a_{k-2,k-2}^{(k-3)}} \right]$$

$$\div \; a_{k-1,k-1}^{(k-2)}$$

$$= \frac{\left(a_{k,k-1}^{(k-2)} a_{kk}^{(k-2)} - a_{k-1,k}^{(k-2)} a_{k,k-1}^{(k-2)}\right) \left(a_{k-1,k-1}^{(k-2)} a_{ij}^{(k-2)}\right)}{a_{k-2,k-2}^{(k-3)} a_{k-2,k-2}^{(k-3)}} \Big/ a_{k-1,k-1}^{(k-2)}$$

$$- \frac{\left(a_{k-1,k-1}^{(k-2)} a_{ik}^{(k-2)} - a_{k-1,k}^{(k-2)} a_{i,k-1}^{(k-2)}\right) \left(a_{k-1,k-1}^{(k-2)} a_{kj}^{(k-2)}\right)}{a_{k-2,k-2}^{(k-3)} a_{k-2,k-2}^{(k-3)}} \Big/ a_{k-1,k-1}^{(k-2)}$$

$$+ \frac{\left(a_{k,k-1}^{(k-2)} a_{ik}^{(k-2)} - a_{i,k-1}^{(k-2)} a_{kk}^{(k-2)}\right) \left(a_{k-1,k-1}^{(k-2)} a_{k-1,j}^{(k-2)}\right)}{a_{k-2,k-2}^{(k-3)} a_{k-2,k-2}^{(k-3)}} \Big/ a_{k-1,k-1}^{(k-2)}$$

$$- \frac{\left(a_{k-1,k}^{(k-2)} a_{k,k-1}^{(k-2)} a_{k-1,j}^{(k-2)} a_{i,k-1}^{(k-2)} - a_{k-1,k}^{(k-2)} a_{k,k-1}^{(k-2)} a_{k-1,j}^{(k-2)} a_{i,k-1}^{(k-2)}\right)}{a_{k-2,k-2}^{(k-3)} a_{k-2,k-2}^{(k-3)}}$$

$$\div \; a_{k-1,k-1}^{(k-2)} \quad .$$

Dividing by $a_{k-1,k-1}^{(k-2)}$ we see that:

$$a_{ij}^{(k)} = (a_{ij}^{(k-2)} c_0^{(k)} + a_{kj}^{(k-2)} c_{i1}^{(k)} + a_{k-1,j}^{(k-2)} c_{i2}^{(k)}) \;/\; a_{k-2,k-2}^{(k-3)}$$

where

$$c_0^{(k)} = (a_{k-1,k-1}^{(k-2)} a_{kk}^{(k-2)} - a_{k-1,k}^{(k-2)} a_{k,k-1}^{(k-2)}) \;/\; a_{k-2,k-2}^{(k-3)}$$

$$c_{i1}^{(k)} = (a_{k-1,k}^{(k-2)} a_{i,k-1}^{(k-2)} - a_{k-1,k-1}^{(k-2)} a_{ik}^{(k-2)}) \;/\; a_{k-2,k-2}^{(k-3)}$$

$$c_{i2}^{(k)} = (a_{k,k-1}^{(k-2)} a_{ik}^{(k-2)} - a_{kk}^{(k-2)} a_{i,k-1}^{(k-2)}) \;/\; a_{k-2,k-2}^{(k-3)} \quad .$$

If n is even, an extra pivot row is calculated at the end. Noting that $c_0^{(k)}$ is calculated once per iteration and $c_{i1}^{(k)}$ and $c_{i2}^{(k)}$ once per row, the two-step algorithm, then, is as follows:

$a_{00}^{(-1)} = 1$;

If n odd then $q = n-1$

  else $q = n-2$;

For $k=2,4,\ldots,q$;

  Compute $c_0^{(k)}$;

  For $i=k+1,k+2,\ldots,n$;

    Compute $c_{i1}^{(k)}$, $c_{i2}^{(k)}$;

    For $j=k+1,k+2,\ldots,n+1$;

$$\text{Do} \quad a_{ij}^{(k)} = (a_{ij}^{(k-2)} c_0^{(k)} + a_{kj}^{(k-2)} c_{i1}^{(k)}$$

$$+ a_{k-1,j} c_{i2}^{(k)}) \,/\, a_{k-2,k-2}^{(k-3)} \;;$$

$$a_{kk}^{(k-1)} = c_0^{(k)} \quad ;$$

For $j=k+1,k+2,\ldots,n+1$;

$$\text{Do} \quad a_{kj}^{(k-1)} = (a_{k-1,k-1}^{(k-2)} a_{kj}^{(k-2)} - a_{k-1,j}^{(k-2)} a_{k,k-1}^{(k-2)})$$

$$\div\, a_{k-2,k-2}^{(k-3)} \;;$$

If n even then

For $j=n,n+1$;

$$\text{Do} \quad a_{nj}^{(n-1)} = (a_{n-1,n-1}^{(n-2)} a_{nj}^{(n-2)} - a_{n-1,j}^{(n-2)} a_{n,n-1}^{(n-2)})$$

$$\div\, a_{n-2,n-2}^{(n-3)} \;.$$

As noted by Lipson [20] and Bareiss [3,4], back-substitution can also be carried out with a fraction-free algorithm. In general, the components of the solution vector x are not integers (or polynomials). However, by Cramer's rule $x = y/d$ where d = determinant of A and $y = A^{adjoint} b$. Since $d = a_{nn}^{(n-1)}$ (see Lipson [20] for proof), the fraction-free back-substitution algorithm is:

$$y_n = a_{n,n+1}^{(n-1} \;; \qquad (2.1.1)$$

For $k=n-1,n-2,\ldots,1$;

$$\text{Do} \quad y_k = \frac{1}{a_{kk}^{(k-1)}} \left(a_{k,n+1}^{(k-1)} a_{nn}^{(n-1)} - \sum_{j=k+1}^{n} \left(a_{kj}^{(k-1)} y_j\right)\right) .$$

Then $x_k = y_k / a_{nn}^{(n-1)}$. If a reduction to lowest terms is desired, a greatest common divisor algorithm must be used.

Multi-step algorithms which perform more than two steps in one iteration have been investigated by Bareiss and Mazukelli [5]. In general, a k-step algorithm for the solution of integer systems uses $(\frac{k+2}{3k})$ times as many multiplications/divisions as the one-step algorithm; the number of additions/subtractions does not change significantly. In section 2.2, for example, we see that the two-step algorithm requires two-thirds as many multiplications/divisions as the one-step algorithm. As the step size grows, however, the amount of computation saved decreases while the pivoting algorithm needed to ensure that all divisors are non-zero becomes increasingly complex. In the following sections, only the one and two-step algorithms will be considered.

2.2 Integer Systems

2.2.1 Growth of Coefficients

Before the number of operations required to perform the one and two-step algorithms can be found, it is necessary to determine the size of the integers produced

at each step.

It has been proved (Lipson [20], Bareiss [3,4]) that each $a_{ij}^{(k-1)}$ is the determinant of a k'th order sub-matrix of $[A,b]$. If each element $a_{ij}^{(0)}$ of $[A,b]$ is an integer of precision $\approx s$, and the maximum value of a single-precision integer is $(m-1)$, then $a_{ij}^{(0)}$ can be written in the polynomial form $\sum_{r=0}^{s-1} c_r^{(ij)} m^r$. If $c_{s-1}^{(ij)} \le c$ for $1 \le i$, $j \le n$, then $a_{ij}^{(0)} < (c+1)m^{s-1}$ and by Hadamard's inequality we have:

$$
\begin{aligned}
|a_{ij}^{(k-1)}| &\le \prod_{1 \le i \le k} \Big[\sum_{1 \le j \le k} (a_{ij}^{(0)})^2 \Big]^{1/2} \\
&< \prod_{1 \le i \le k} \Big[\sum_{1 \le j \le k} ((c+1)m^{s-1})^2 \Big]^{1/2} \\
&= \prod_{1 \le i \le k} \big[k(c+1)^2 (m^{s-1})^2 \big]^{1/2} \\
&= \prod_{1 \le i \le k} \big[\sqrt{k}\,(c+1)m^{s-1} \big] \\
&= \big[\sqrt{k}\,(c+1)m^{s-1} \big]^k \\
&= (m^w)^k
\end{aligned}
$$

where $m^w = \sqrt{k}(c+1)m^{s-1}$. Then $a_{ij}^{(k-1)}$ has precision $\approx wk$ where

$$
\begin{aligned}
w &= \log_m(\sqrt{k}(c+1)m^{s-1}) \\
&= \tfrac{1}{2}\log_m k + \log_m(c+1) + (s-1) \qquad (2.2.1)
\end{aligned}
$$

$$< \frac{1}{2} \log_m n + \log_m (c+1) + (s-1) \quad .$$

If each $a_{ij}^{(0)}$ is single-precision, and if

$$\frac{1}{2} \log_m n + \log_m (c+1) \approx 1 \qquad (2.2.2)$$

then $a_{ij}^{(k-1)}$ has precision $\approx k$.

2.2.2 One-Step Elimination

In this section it is assumed that 'classical' algorithms for performing multi-precision arithmetic are used.

Let $C_\times[k;j]$ represent the number of single precision operations required to multiply a k-precision integer by a j-precision integer, and let $C_\div$ and C_+ be similarly defined. Let A represent single-precision addition or subtraction operations, and let M represent single-precision multiplication or division operations.

Then using the classical algorithms:

$$C_+[k;k] = [2k]A$$

$$C_\times[k;j] = [kj]M + [2kj]A$$

$$C_\div[k;j] = [(j+2)(k-j+1)]M + [3(j+1)(k-j+1)]A \quad .$$

If (2.2.2) is satisfied and hence $a_{ij}^{(k-1)}$ has

precision $= k$, then triangularization by the one-step algorithm requires the following operations:

$$\sum_{k=1}^{n-1} (n-k)(n-k+1)(2C_{\times}[k;k] + C_{+}[2k;2k] + C_{\div}[2k;k-1])$$

$$= \sum_{k=1}^{n-1} (n-k)(n-k+1)([2k^2]M + [4k^2]A + [4k]A +$$

$$+ [(k+1)(k+2)]M + [3k(k+2)]A)$$

$$= \sum_{k=1}^{n-1} [3k^2-k^3(6n)+k^2(3n^2-3n-1)+k(3n^2-n-2)+2n^2+2n]M$$

$$+ \sum_{k=1}^{n-1} [7k^2+k^3(-14n+3)+k^2(7n^2-13n-10)+k(10n^2+10n)]A$$

$$= [\frac{1}{10} n^5 + \frac{1}{2} n^4 + \frac{7}{6} n^3 - \frac{1}{2} n^2 - \frac{19}{15}]M$$

$$+ [\frac{7}{30} n^5 + \frac{17}{12} n^4 + \frac{5}{3} n^3 - \frac{17}{12} n^2 - \frac{19}{10} n]A \quad .$$

Back-substitution by algorithm (2.1.1) requires:

$$\sum_{k=1}^{n-1} \{(n-k+1)C_{\times}[n;k] + (n-k)C_{+}[n+k;n+k] + C_{\div}[n+k;k]\}$$

$$= \sum_{k=1}^{n-1} \{[(n-k+1)(nk)]M + [(n-k+1)(2nk)]A + [(n-k)(2)(n+k)]A$$

$$+ [(k+2)(n+1)]M + [3(k+1)(n+1)]A\}$$

$$= [\frac{1}{6} n^4 + n^3 + \frac{4}{3} n^2 - \frac{1}{2} n - 2]M$$

$$+ [\frac{1}{3} n^4 + \frac{23}{6} n^3 + \frac{11}{3} n^2 - \frac{11}{6} n - 6]A \quad .$$

The total operations required then, for both triangularization and back-substitution are:

$$[\frac{1}{10} n^5 + \frac{2}{3} n^4 + \frac{13}{6} n^3 + \frac{5}{6} n^2 - \frac{53}{60} n - 2]M$$
$$+ [\frac{7}{30} n^5 + \frac{7}{4} n^4 + \frac{11}{2} n^3 + \frac{9}{4} n^2 - \frac{56}{15}]A \quad .$$

If inequality (2.2.2) is not satisfied, then $a_{ij}^{(k-1)}$ has precision $\approx wk$ where w is defined by equation (2.2.1) and triangularization requires:

$$\sum_{k=1}^{n-1} (n-k)(n-k+1)(2C_{\times}[wk;wk] + C_{+}[2wk;2wk] + C_{\div}[2wk;wk-w])$$

$$= \sum_{k=1}^{n-1} (n-1)(n-k+1)([2w^2k^2]M + [4w^2k^2]A + [4wk]A$$
$$+ [(wk-w+2)(wk+w+1)]M + [3(wk-w+1)(wk+w+1)]A)$$

$$\approx \sum_{k=1}^{n-1} (n-k)^2([3w^2k^2]M + [7w^2k^2]A$$

$$= [\frac{1}{10} n^5w^2]M + [\frac{7}{30} n^5w^2]A \quad .$$

The number of operations, then, is increased by a factor of w^2 if $w > 1$, and the order of complexity of the one-step method is n^5w^2 .

2.2.3 <u>Two-Step Elimination</u>

If inequality (2.2.2) is satisfied and $a_{ij}^{(k-1)}$ has

precision $\approx k$, then triangularization by the two-step algorithm requires the following operations:

$$\sum_{k=2,4,\ldots}^{q} \{2C_{\times}[k-1;k-1] + C_{+}[2k-2;2k-2] + C_{\div}[2k-2;k-2]$$

$$+ (n-k)(2)(2C_{\times}[k-1;k-1] + C_{+}[2k-2;2k-2] + C_{\div}[2k-2;k-2])$$

$$+ (n-k)(n-k+1)(3C_{\times}[k-1;k] + 2C_{+}[2k-1;2k-1] + C_{\div}[2k-1;k-2])$$

$$+ (n-k+1)(2C_{\times}[k-1;k-1] + C_{+}[2k-2;2k-2] + C_{\div}[2k-2;k-2])\}$$

$$+ \begin{cases} 0 , & n \text{ odd} \\ 2C_{\times}[n-1;n-1] + C_{+}[2n-2;2n-2] + C_{\div}[2n-2;n-2] , & n \text{ even} \end{cases}$$

where $q = \begin{cases} n-1 , & n \text{ odd} \\ n-2 , & n \text{ even} \end{cases}$

$$= \sum_{k=2,4,\ldots}^{q} \{[4k^2+k^3(-8n-14)+k^2(4n^2+15n+16)+k(-n^2-10n-12)$$

$$+ 6n + 4]M$$

$$+ [9k + k^3(-18n-25) + k^2(9n^2+20n+11)$$

$$+ k(5n^2+13n+11) - 10n^2 - 19n - 6]A\}$$

$$+ \begin{cases} 0 , & n \text{ odd} \\ [6n^2-6n+4]M + [14n^2-8n-6]A , & n \text{ even} \end{cases}$$

$$= \sum_{j=1}^{q/2} \{[4(2j)^4 + (2j)^3(-8n-14) + (2j)^2(4n^2+15n+16)$$

$$+ (2j)(-n^2-10n-12) + 6n + 4]M$$

$$+ [9(2j)^4 + (2j)^3(-18n-25) + (2j)^2(9n^2+20n+11)$$

$$+ (2j)(5n^2+13n+11) - 10n^2 - 19n - 6]A\}$$

$$+ \begin{cases} 0 \,, & n \text{ odd} \\ [6n^2-6n+4]M + [14n^2-8n-6]A & , n \text{ even} \end{cases}$$

$$= \begin{cases} [\frac{1}{15} n^5 + \frac{1}{2} n^4 + \frac{1}{6} n^3 + \frac{5}{4} n^2 - \frac{37}{30} n - \frac{3}{4}]M \,, & n \text{ odd} \\ [\frac{1}{15} n^5 + \frac{1}{2} n^4 + \frac{1}{6} n^3 + n^2 + \frac{4}{15} n]M \,, & n \text{ even} \end{cases}$$

$$= \begin{cases} [\frac{3}{20} n^5 + \frac{145}{48} n^4 + \frac{19}{3} n^3 + \frac{223}{24} n^2 + \frac{451}{60} n - \frac{61}{16}]A \,, & n \text{ odd} \\ [\frac{3}{20} n^5 + \frac{145}{48} n^4 + \frac{1}{12} n^3 - \frac{7}{12} n^2 + \frac{113}{30} n - 10]A \,, & n \text{ even} \end{cases}$$

Back-substitution, as in the one-step case, requires:

$$[\frac{1}{6} n^4 + n^3 + \frac{4}{3} n^2 - \frac{1}{2} n - 2]M$$

$$+ [\frac{1}{3} n^4 + \frac{23}{6} n^3 + \frac{11}{3} n^2 - \frac{11}{6} n - 6]A \;.$$

The total number of operations required is:

$$\leq [\frac{1}{15} n^5 + \frac{2}{3} n^4 + \frac{7}{6} n^3 + \frac{31}{12} n^2 - \frac{26}{15} n - \frac{11}{4}]M$$

$$+ [\frac{3}{20} n^5 + \frac{161}{48} n^4 + \frac{61}{6} n^3 + \frac{311}{24} n^2 + \frac{341}{60} n - \frac{35}{16}]A \;.$$

Comparing this total with that obtained for the one-step algorithm, it can be seen that M has decreased by approximately one-third, while A has decreased by one-

sixtieth. The order of complexity is still, however, n^5.

If $a_{ij}^{(k-1)}$ has precision wk where $w > 1$, then again the order of complexity is increased by a factor of w^2.

2.3 Polynomial Systems

2.3.1 Growth of Coefficients

If the elements $a_{ij}^{(0)}$ of the augmented matrix $[A,b]$ are polynomials of degree d in each of r variables x_i, having single-precision coefficients $\approx c$, then each term of $a_{ij}^{(0)} \approx c(x_1^d \ldots x_r^d)$. Then $a_{ij}^{(k-1)}$ is a k'th order determinant and by Hadamard's inequality:

$$\left|\text{term of } a_{ij}^{(k-1)}\right| \leq \prod_{1 \leq i \leq k} \left[\sum_{1 \leq j \leq k} c^2 (x_1^d x_2^d \ldots x_r^d)^2 \right]^{1/2}$$

$$= \prod_{1 \leq i \leq k} [\sqrt{k}\, c(x_1^d x_2^d \ldots x_r^d)]$$

$$= [\sqrt{k}\, c]^k \, (x_1^{dk} x_2^{dk} \ldots x_r^{dk})$$

$$= (m^w)^k \, (x_1^{dk} x_2^{dk} \ldots x_r^{dk}) \quad .$$

Then $a_{ij}^{(k-1)}$ is a polynomial of degree $\approx kd$ in each of r variables, with coefficients of precision $\leq wk$ where

$$w = \log_m (\sqrt{k}\, c) \qquad (2.3.1)$$

$$= \frac{1}{2} \log_m k + \log_m c$$

$$\leq \frac{1}{2} \log_m n + \log_m c$$

and $(m-1)$ is the maximum value of a single-precision integer.

2.3.2 One-Step Elimination

As in section 2.2, the assumption is made that the classical algorithms are used for polynomial arithmetic.

Let $C_\times[(d)_k^r;(e)_j^r]$ represent the number of operations required to multiply a polynomial with k-precision coefficients and degree d in each of r variables by a polynomial with j-precision coefficients and degree e in each of r variables. Let $C_+[(d)_k^r;(e)_j^r]$ and $C_\div[(d)_k^r;(e)_j^r]$ be similarly defined.

Then

$$C_+[(d)_k^r;(d)_k^r] = (d+1)^r C_+[k;k] = [(d+1)^r(2k)]A$$

$$\begin{aligned} C_\times[(d)_k^r;(e)_j^r] &= (d+1)^r(e+1)^r C_\times[k;j] \\ &\quad + (d+1)^r(e+1)^r C_+[k+j;k+j] \\ &= [(d+1)^r(e+1)^r(kj)]M \\ &\quad + [(d+1)^r(e+1)^r(2kj+2k+2j)]A \end{aligned}$$

$$\begin{aligned} C_\div[(d)_k^r;(e)_j^r] &= (d-e+1)^r C_\div[k;j] \\ &\quad + (d-e+1)^r((e+1)^r-1)C_\times[k-j;j] \\ &\quad + (d-e+1)^r(e+1)^r C_+[k;k] \end{aligned}$$

$$= [(d-e+1)^r(2k-j+2+(e+1)^r(k-j)(j))]M$$
$$+ [(d-e+1)^r(kj-j^2+3k+3$$
$$+ (e+1)^2(2)(kj-j^2+k))]A \quad .$$

Triangularization by the one-step algorithm, then, requires the following operations:

$$\sum_{k=1}^{n-1} (n-k)(n-k+1)(2C_\times[(kd)^r_{wk};(kd)^r_{wk}] + C_+[(2kd)^r_{2wk};(2kd)^r_{2wk}]$$
$$+ C_\div[(2kd)^r_{2wk};(kd-d)^r_{wk-w}])$$

$$= \sum_{k=1}^{n-1} (n-k)(n-k+1)\{[(2)(kd+1)^{2r}(w^2k^2)]M$$

$$+ [(2)(kd+1)^r(2w^2k^2+4wk)]A + [(2kd+1)^r(4pk)]A$$

$$+ [(kd+d+1)^r(3wk+w+2+(kd-d+1)(wk+w)(wk-w))]M$$

$$+ [(kd+d+1)^r(w^2k^2+4wk+2w^2k-w^2+3+(kd-d+1)^r$$
$$(2)(w^2k^2-w^2)]A\}$$

$$\simeq \sum_{k=1}^{n-1} (n-k)^2\{[2w^2k^{2r+2}d^{2r} + w^2k^{2r+2}d^{2r}]M$$

$$+ [4w^2k^{2r+2}d^{2r} + 2w^2k^{2r+2}d^{2r}]A\}$$

(taking terms of highest order only)

$$= \sum_{k=1}^{n-1} (n-k)^2\{[3w^2k^{2r+2}d^{2r}]M + [6w^2k^{2r+2}d^{2r}]A\}$$

$$= [3w^2d^{2r} \sum_{k=1}^{n-1} (k^{2r+4} - 2nk^{2r+3} + n^2k^{2r+2})]M$$

$$+ [6w^2d^{2r} \sum_{k=1}^{n-1} (k^{2r+4} - 2nk^{2r+3} + nk^{2r+2})]A$$

$$\simeq [3w^2d^{2r} (\frac{n^{2r+5}}{2r+5} - \frac{(2n)(n^{2r+4})}{2r+4} + \frac{(n^2)(n^{2r+3})}{2r+3})]M$$

$$+ [6w^2d^{2r} (\frac{n^{2r+5}}{2r+5} - \frac{(2n)(n^{2r+4})}{2r+4} + \frac{(n^2)(n^{2r+3})}{2r+3})]A$$

$$= [w^2n^{2r+5}d^{2r} (\frac{3}{2r+5} - \frac{6}{2r+4} + \frac{3}{2r+3})]M$$

$$+ [w^2n^{2r+5}d^{2r} (\frac{6}{2r+5} - \frac{12}{2r+4} + \frac{6}{2r+3})]A \quad .$$

The operations needed for back-substitution are:

$$\sum_{k=1}^{n-1} \{(n-k+1)C_\times[(nd)^r_{wn};(kd)^r_{wk}]$$

$$+ (n-k)C_+[(nd+kd)^r_{wn+wk};(nd+kd)^r_{wn+wk}]$$

$$+ C_\div[(nd+kd)^r_{wn+wk};(kd)^r_{wk}]\}$$

$$= \sum_{k=1}^{n-1} \{[(n-k+1)(nd+1)^r(kd+1)^r(w^2nk)]M$$

$$+ [(n-k+1)(nd+1)^r(kd+1)^r(2w^2+2wn+2wk)]A$$

$$+ [(n-k)(nd+kd+1)^r(2wn+2wk)]A$$

$$+ [(nd+1)^r(2wn+wk+2+(kd+1)^r(wn)(wk)]M$$

$$+ [(nd+1)^r(w^2nk+3wn+3wk+3+(kd+1)^r(2)(w^2nk+wn+wk)]A\}$$

$$\approx [w^2 n^{r+4} d^{2r} (\frac{1}{r+2} - \frac{1}{r+3})]M + [w^2 n^{r+4} d^{2r} (\frac{2}{r+2} - \frac{2}{r+3})]A \quad .$$

The order of complexity for the one-step algorithm, then, is $w^2 n^{2r+5} d^{2r}$. For univariate polynomials, the one-step algorithm requires approximately $\frac{1}{35} w^2 n^7 d^2$ multiplications/divisions and $\frac{3}{35} w^2 n^7 d^2$ additions/subtractions.

If the polynomials $a_{ij}^{(0)}$ are of low degree and have small coefficients, and n is small, so that $wn \leq 1$, then all coefficients remain single precision. In this case the operations required are

$$[n^{2r+3} d^{2r} (\frac{3}{2r+3} - \frac{6}{2r+2} + \frac{3}{2r+1})]M$$

$$+ [n^{2r+3} d^{2r} (\frac{3}{2r+3} - \frac{6}{2r+2} + \frac{3}{2r+1})]A \quad .$$

2.3.3 Two-Step Elimination

If the two-step algorithm is used to triangularize $[A,b] = (a_{ij}^{(0)})$ where $a_{ij}^{(0)}$ is a polynomial with single-precision coefficients and degree d in each of r variables, then the operations required are:

$$\sum_{k=2,4,\ldots}^{q} \{2C_{\times}[(kd-d)^r_{wk-w};(kd-d)^r_{wk-w}]$$

$$+ C_{+}[(2kd-2d)^r_{2wk-2w};(2kd-2d)^r_{2wk-2w}]$$

$$+ C_{\div}[(2kd-2d)^r_{2wk-2w};(kd-2d)^r_{wk-2w}]$$

$$
\begin{aligned}
&+ (n-k)(2)(2C_{\times}[(kd-d)^{r}_{wk-w};(kd-d)^{r}_{wk-w}] \\
&\qquad + C_{+}[(2kd-2d)^{r}_{2wk-2w};(2kd-2d)^{r}_{2wk-2w}] \\
&\qquad + C_{\div}[(2kd-2d)^{r}_{2wk-2w};(kd-2d)^{r}_{wd-2w}]) \\
&+ (n-k)(n-k+1)(3C_{\times}[(kd-d)^{r}_{wk-w};(kd)^{r}_{wk}] \\
&\qquad + 2C_{+}[(2kd-d)^{r}_{2wk-w};(2kd-d)^{r}_{2wk-w}] \\
&\qquad + C_{\div}[(2kd-d)^{r}_{2wk-w};(kd-2d)^{r}_{wk-2w}]) \\
&+ (n-k+1)(2C_{\times}[(kd-d)^{r}_{wk-w};(kd-d)^{r}_{wk-w}] \\
&\qquad + C_{+}[(2kd-2d)^{r}_{2wk-2w};(2kd-2d)^{r}_{2wk-2w}] \\
&\qquad + C_{\div}[(2kd-2d)^{r}_{2wk-2w};(kd-2d)^{r}_{wk-2w}]\} \\
&+ \begin{cases} 0 \\ 2C_{\times}[(nd-d)^{r}_{wn-w};(nd-d)^{r}_{wn-w}] \end{cases} \\
&\qquad + \begin{cases} 0 \\ C_{+}[(2nd-2d)^{r}_{2wn-2w};(2nd-2d)^{r}_{2wn-2w}] \end{cases} \\
&\qquad + \begin{cases} 0\,, & n \text{ odd} \\ C_{\div}[(2nd-2d)^{r}_{2wn-2w};(nd-2d)^{r}_{wn-2w}]\,, & n \text{ even} \end{cases}
\end{aligned}
$$

where $q = \begin{cases} n-1\,, & n \text{ odd} \\ n-2\,, & n \text{ even} \end{cases}$

$$
\approx \sum_{k=2,4,\ldots}^{q} \{(n-k)^{2}[4w^{2}k^{2r+2}d^{2r}]M + [8w^{2}k^{2r+2}d^{2r+2}]A\}
$$

(taking terms of highest order)

$$\approx [w^2 n^{2r+5} d^{2r} (\frac{2}{2r+5} - \frac{4}{2r+4} + \frac{2}{2r+3})]M$$

$$+ [w^2 n^{2r+5} d^{2r} (\frac{4}{2r+5} - \frac{8}{2r+4} + \frac{4}{2r+5})]A \quad .$$

Back-substitution is performed as in the one-step algorithm.

The order of complexity of the two-step algorithm is the same as that of the one-step algorithm - $w^2 n^{2r+5} d^{2r}$. However, the coefficient is smaller. In the case of univariate polynomials, for example, M is approximately $\frac{2}{105} w^2 n^7 d^2$ and A is approximately $\frac{4}{105} w^2 n^7 d^5$. These totals are each one-third smaller than those obtained for the one-step algorithm.

A similar reduction in the operations required can be observed for systems which satisfy $wn \leq 1$. For such systems the two-step algorithm requires:

$$[n^{2r+3} d^{2r} (\frac{2}{2r+3} - \frac{4}{2r+2} + \frac{2}{2r+1})]M$$

$$+ [n^{2r+3} d^{2r} (\frac{2}{2r+3} - \frac{4}{2r+2} + \frac{2}{2r+1})]A \quad .$$

For univariate polynomials, these totals are $[\frac{1}{15} n^5 d^2]M$ and $[\frac{1}{15} n^5 d^2]A$, as opposed to the totals of $[\frac{1}{10} n^5 d^2]M$ and $[\frac{1}{10} n^5 d^2]A$ obtained for the one-step algorithms.

If the polynomial coefficients of A and b have varying degrees d_i , $1 \leq i \leq r$, instead of uniform degree

$\bar{d}$, then the operations required are:

$$[w^2 n^{2r+5} (\prod_{i=1}^{r} d_i)(\frac{2}{2r+5} - \frac{4}{2r+4} + \frac{2}{2r+3})]M$$

$$+ [w^2 n^{2r+5} (\prod_{i=1}^{r} d_i)(\frac{4}{2r+5} - \frac{8}{2r+4} + \frac{4}{2r+3})]A \quad .$$

2.4 Fast Arithmetic

Bareiss [3,4] has suggested that 'fast' algorithms for the multiplication of multi-precision numbers can be used to reduce the order of complexity of the multi-step methods. He concludes that with such techniques, the multi-step algorithms use $O(wn^4)$ operations in the integer case. To the contrary, in this section it is shown that fast techniques produce an algorithm which is, at best, $O(wn^4 \log n)$.

Several algorithms which multiply two k-precision numbers in less than the classical ck^2 operations have been discovered. The first of these was an $O(k^{\log_2 3})$ algorithm suggested by A. Karatsuba in 1962. Shortly afterwards A.L. Toom produced a computer-circuitry scheme to multiply in $O(ck^{1+3.5/\sqrt{\log_2 k}})$ operations, and in 1966 S.A. Cook adapted the method to computer programs. A. Schönhage (1965) used modular techniques to obtain an $O(ck^{1+\sqrt{2\log_2 k}}(\log_2 k)^{3/2})$ method. For a description and analysis of these algorithms, see Knuth ([19], Chapter

4.3.3, "How Fast Can We Multiply?").

As Bareiss points out, Schönhage has speculated that at best, an $O(k^{1+\varepsilon})$ algorithm is possible, where $\varepsilon = \log\log k/\log k$. But

$$\begin{aligned} k^{1+\varepsilon} &= k^{1+\log\log k/\log k} \\ &= k.k^{\log\log k/\log k} \\ &= k(2^{\log k})^{\log\log k/\log k} \\ &= k.2^{\log\log k} \\ &= k\log k \quad . \end{aligned}$$

Schönhange, then, is suggesting an optimum of $ck \log k$ operations to perform k-precision multiplication.

None of the algorithms mentioned above achieves this optimum. In fact Knuth [19] has shown that the best of these is a modification of the Toom-Cook algorithm which is $O(k.2^{\sqrt{2\log_2 k}}\log_2 k)$. Another algorithm discussed by Borodin and Munro [9] is based on the Fast Fourier Transform technique and uses $O(k\log^2 k)$ operations.

If Schönhage's optimum of $k \log k$ is assumed and a similar result for division is assumed, the operations required for the one-step algorithm are:

$$\sum_{k=1}^{n-1} (n-k)(n-k+1)([2wk \log wk]M + [2wk \log wk]A$$
$$+ [4wk]A + [2wk \log 2wk]M + [2wk \log 2wk]A)$$

$$\approx \sum_{k=1}^{n-1} (n-k)^2 (4wk \log wk]M + [4wk \log wk]A + [4wk]A)$$

$$\approx [4w \sum_{k=1}^{n-1} (n^2k \log k - 2nk^2 \log k + k^3 \log k)]M$$
$$+ [4w \sum_{k=1}^{n-1} (n^2k \log k - 2nk^2 \log k + k^3 \log k)]A$$
$$+ [4w \sum_{k=1}^{n-1} k]A \quad .$$

Since

$$\sum_{k=1}^{n-1} k \log k \approx \frac{1}{2} n^2 \log n - \frac{1}{4} n^2 - \frac{1}{2} n \log n + \frac{1}{4} \quad ,$$

$$\sum_{k=1}^{n-1} k^2 \log k \approx \frac{1}{3} n^3 \log n - \frac{1}{9} n^3 - \frac{1}{2} n^2 \log n + \frac{1}{9} \quad ,$$

$$\sum_{k=1}^{n-1} k^3 \log k \approx \frac{1}{4} n^4 \log n - \frac{1}{16} n^4 - \frac{1}{2} n^3 \log n + \frac{1}{16} \quad ,$$

the operations required are approximately:

$$[\frac{1}{3} wn^4 \log n - \frac{13}{36} wn^4 + wn^2 - \frac{8}{9} wn + \frac{1}{4}]M$$
$$+ [\frac{1}{3} wn^4 \log n - \frac{13}{36} wn^4 + 3wn^2 - \frac{26}{9} wn + \frac{1}{4}]A \quad .$$

This total constrasts with the conclusion of Bareiss [3,4]. He says that the number of multiplications needed for the one-step method, assuming calculation of

$a_{ij}^{(k)}$ requires one multiplication (he adjusts later for the two multiplications needed and ignores the division), is $\sum_{k=1}^{n-1} (n-k)^2 (k \frac{d_\kappa}{\omega})^{1+\varepsilon}$ where kd_κ/ω is the precision at step k and k-precision multiplication takes $k^{1+\varepsilon}$ single-precision multiplications. Bareiss then states that:

$$\sum_{k=1}^{n-1} (n-k)^2 \, (k \frac{d_\kappa}{\omega})^{1+\varepsilon} \simeq \frac{2n^{4+\varepsilon}(d_n/\omega)^{1+\varepsilon}}{(2+\varepsilon)(3+\varepsilon)(4+\varepsilon)} \xrightarrow[\varepsilon \to 0]{} \frac{n^4}{12} \cdot \frac{d_n}{\omega} \; .$$

This result is incorrect. For taking Schönhage's optimum estimate of $\varepsilon \doteq \log \log n / \log n$, $n^{4+\varepsilon}$ does not behave like n^4 in the limit as $\varepsilon \to 0$ but rather, as already shown, $n^{4+\varepsilon} = n \log n$.

With optimal fast multiplication, then, the one-step algorithm has order $wn^4 \log n$. Even if such an optimal fast multiplication algorithm can be found, it is not clear what the leading coefficient of that method will be. Hence Bareiss' contention that the one-step algorithm has order wn^4 with a coefficient less than one is incorrect. Since the order of complexity of the multi-step algorithms with larger step sizes is the same as that of the one-step algorithm, it can be seen that all of these algorithms are $O(wn^4 \log n)$.

Knuth [19] notes that multiplication of k-bit numbers can be accomplished in k steps "if we leave the

domain of conventional computer programming and allow ourselves to build a computer which has an unlimited number of components all acting at once". Such an algorithm would give $O(wn^4)$ multi-step methods. For conventional computers, however, the multi-step algorithms are of order $wn^4 \log n$, at best.

CHAPTER THREE

Congruential Methods

The congruential method of solving a system of linear equations with integer coefficients was first proposed in 1961 by Takahasi and Ishibashi [27]. Since that time Newman [24], Borosh and Fraenkel [10], Howell and Gregory [16,17,18], Cabay [13], and Bareiss [3,4] have suggested improvements. Some attention has also been given, notably by McClellan [23], to the case of polynomial coefficients.

To solve a system of linear equations $Ax = b$ with integer (or polynomial) coefficients, the first step in the procedure is to solve the system by Gaussian elimination modulo a number of primes $p_1, p_2, \ldots, p_t$. The true solution is then constructed from the t modular solutions by means of the Chinese Remainder Theorem.

In general, the components of the solution vector x are not integers (or polynomials). By Cramer's rule, however, $x = y/d$, where d = determinant of A and $y = A^{\text{adjoint}}b$ are integer (or polynomial) quantities. The method, then, is to compute for each k , $1 \leq k \leq t$, the solution $(\bar{d}_k, \bar{y}_k)$ of

$$(*) \qquad \bar{A}_k \bar{y}_k \equiv \bar{d}_k \bar{b}_k \pmod{p_k}$$

where $\bar{A}_k = A \pmod{p_k}$, $\bar{b}_k = b \pmod{p_k}$ and

$$(**) \qquad \begin{cases} \bar{d}_k = d \pmod{p_k} \\ \bar{y}_k = y \pmod{p_k} \end{cases} .$$

The unique integer (or polynomial) d and the unique vector of integers (or polynomials) y satisfying (**) can then be constructed by means of the Chinese Remainder Theorem if the system (*) has been solved for a sufficient number of primes p_k .

The congruential method of solving a system of linear equations with integer coefficients is discussed in section 3.1 and detailed operation counts for the method are given. In section 3.2, systems with polynomial coefficients are considered and approximate operation counts are found.

The congruential method is often hampered by the occurrence of what has come to be known as 'bad' primes. Recently it has been shown that these primes are not 'bad' in the case of systems with integer coefficients. For systems with polynomial coefficients, it is shown that one type of prime is not 'bad', that subject to a certain natural condition, a second type need not occur, and that the third type can be used with little or no loss of computation.

Also in section 3.2.1, a theorem for efficiently

terminating the Chinese Remainder formula for polynomials is given.

3.1 Integer Systems

3.1.1 The Method

When a system of linear equations $Ax = b$ with integer coefficients is solved by the congruential method, the system is solved modulo t primes p_k , $1 \le k \le t$. Suppose that the p_k have been selected. For each prime p_k the quantity $(\bar{d}_k, \bar{y}_k)$ satisfying

$$\bar{d}_k \equiv d \pmod{p_k}$$

$$\bar{y}_k \equiv y \pmod{p_k}$$

is determined by solving the system $Ax = b$ using Gaussian elimination modulo p_k with partial pivoting. The algorithm for triangularization becomes:

For $h = 1,2,\ldots,n-1$;
  For $i = h+1,h+2,\ldots,n$;
    For $j = h+1,h+2,\ldots,n+1$;
      Do $a_{ij}^{(h)} \equiv a_{ij}^{(h-1)} - a_{ih}^{(h-1)}(a_{hh}^{(h-1)})^{-1}a_{hj}^{(h-1)} \pmod{p_k}$.

By partial pivoting, we mean that if at any stage the diagonal pivot element $a_{hh}^{(h-1)}$ becomes zero, an attempt is made to rearrange the remainding rows to obtain a non-

zero pivot. The determinant of $A \pmod{p_k}$ is immediately available as the product of the pivot elements. To determine $\bar{y}_k$, back-substitution modulo p_k is first used to produce a vector $\bar{x}_k$ satisfying $A\bar{x}_k \equiv b \pmod{p_k}$. Then $\bar{x}_k$ also satisfies $\bar{x}_k \equiv x \pmod{p_k}$. But since $y = dx$, it follows that

$$\bar{y}_k = \bar{d}_k \bar{x}_k \pmod{p_k} .$$

The conditions which should be imposed on the primes p_k now become apparent. Each p_k should be:

(1) less than or equal to the word-size of the computer so that all modular operations are performed on single-precision integers.

(2) as large as possible (within the restriction imposed by (1)) so that a minimum number of moduli can be used.

(3) prime so that $(a_{hh}^{(h-1)})^{-1} \pmod{p_k}$ is defined whenever $a_{hh}^{(h-1)} \neq 0$.

There are two algorithms available for finding $a^{-1} \pmod{p}$. By Fermat's Theorem, $a^{-1} = a^{p-2} \pmod{p}$. However, Collins [14] has shown that it is about twice as efficient to use the extended Euclidean algorithm, which finds integers a' and p' such that $aa' + pp' = (a,p)$. Since a and p are relatively prime, $aa' + pp' = 1$. Then $aa' = 1 \pmod{p}$ and a' is the required inverse.

The algorithm (see Knuth [19], page 302) is as follows:

$(u_1, u_2) \leftarrow (0, p)$;

$(v_1, v_2) \leftarrow (1, a)$;

While $v_2 \neq 0$ do

$q \leftarrow \lfloor u_2/v_2 \rfloor$

$(t_1, t_2) \leftarrow (u_1, u_2) - (v_1, v_2)q$

$(u_1, u_2) \leftarrow (v_1, v_2)$

$(v_1, v_2) \leftarrow (t_1, t_2)$;

$a' = v_1$.

By Lamé's Theorem, this algorithm requires a maximum number of divisions when a and p are consecutive Fibonacci numbers. In this case (see Knuth [19], page 320), the number of divisions $\leq \lceil 4.785 + \log_{10} a + 1.672 \rceil - 2 < 5 \log_{10} p$. The average number of divisions required, however, is $\frac{7}{12} \log_2 p$ (Collins [14]) and it is this latter estimate which will be used.

It may happen that during the h'th step of the triangularization process a zero pivot element is encountered. (That is, no non-zero pivot can be obtained by rearranging the rows of the matrix.) In this case $(a_{hh}^{(h-1)})^{-1} \pmod{p_k}$ no longer exists; however, it is now known that $\overline{d}_k = 0$. Then either $d = 0$ (in which case the coefficient matrix A is singular and no solution exists), or d is a multiple of p_k . In the latter

case $\bar{y}_k$ can no longer be found from the equation $\bar{y}_k = \bar{x}_k \bar{d}_k$.

Several approaches have been taken to this 'bad' prime problem. The probability that p_k is a factor of d is small. Therefore Takahasi and Ishibashi [27] and Newman [24] suggest terminating the algorithm on the assumption that A is singular. Borosh and Fraenkel [10], on the other hand, discard this prime and use another if any previous or subsequent moduli show that d is not in fact zero.

However, Cabay [13] has shown that $\bar{y}_k$ can be found even when $\bar{d}_k$ is zero. The column with the zero pivot element (column j, say) is ignored and the rest of the columns are eliminated as if column j were not present. Two cases arise. If rank $[A,b] = n \pmod{p_k}$ it can be shown that

$$(\bar{y}_k)_j = (-1)^{n-1} a_{11}^{(0)} \cdots a_{j-1,j-1}^{(j-2)} a_{j,j+1}^{(j-1)} \cdots a_{n-1,n}^{(n-2)} b_n^{(n-2)} ,$$

that $(\bar{y}_k)_i = 0$ for $j+1 \le i \le n$, and that $(\bar{y}_k)_i$, $1 \le i \le j-1$, can be found by back-substitution in $A^{(n-2)} y = 0 \pmod{p_k}$. If rank $[A,b] < n \pmod{p_k}$, then a second zero pivot element is encountered and $\bar{y}_k$ must be the zero vector. A slight modification of the Gaussian elimination algorithm is therefore sufficient to handle a 'bad' prime, should one occur.

From $\bar{d}_k$ and $\bar{y}_k$, $1 \le k \le t$, a solution $y \equiv \bar{y}_k \pmod{p_k}$, $d \equiv \bar{d}_k \pmod{p_k}$ is constructed by means of the Chinese Remainder Theorem for integers. There are two constructive proofs of this theorem: the Newtonian formula and the Lagrangian formula. Lipson [21] has shown that the Newtonian formula is to be preferred in as much as the number of moduli required need not be known in advance, the storange requirements are smaller, and fewer multiplications/divisions need be performed.

The Newtonian formula for constructing an integer I satisfying $I \equiv u_k \pmod{p_k}$ for $1 \le k \le t$ is given by:

$$I = a_1 + a_2p_1 + a_3p_1p_2 + \ldots + a_tp_1p_2\cdots p_{t-1} \qquad (3.1.1)$$

where $a_1 = s_1 = u_1$

$$a_k = (u_k - s_{k-1})(p_1^{-1}p_2^{-1}\cdots p_{k-1}^{-1}) \pmod{p_k}$$

$$s_k = s_{k-1} + a_kp_1p_2\cdots p_{k-1} \quad .$$

It is assumed that $c_k = p_1^{-1}p_2^{-1}\cdots p_{k-1}^{-1} \pmod{p_k}$, $1 \le k \le t$, is pre-computed, again by means of the extended Euclidean algorithm.

Lipson [21] also analysed three implementations of the Newtonian formula. The best of these (from the viewpoint of operations and storage required) proceeds as follows:

$a_1 = u_1$;

For $k = 2$ to t do

$v = a_{k-1}$;

For $i = k-2$ step -1 to 1 do

$v = vp_i + a_i \pmod{p_k}$;

$a_k = (u_k - v) \times c_k \pmod{p_k}$.

Then I is computed by Horner's rule:

$$I = ((\ldots(a_t p_{t-1} + a_{t-1})p_{t-2} + a_{t-2})\ldots + a_2)p_1 + a_1 \ . \quad (3.1.2)$$

Observe that single-precision operations only are required to compute the coefficients a_k .

The integer I so constructed is only one out of infinitely many integers satisfying the congruences

$$(*) \qquad I \equiv u_k \pmod{p_k} \quad 1 \le k \le t \ .$$

We know however, by the Chinese Remainder Theorem, that there exists a unique I satisfying $(*)$ if it is known that $c < I < c + p_1 p_2 \cdots p_t$ where c is an arbitrary integer. Since d and y may be negative, we wish to represent the residues symmetrically about zero. Therefore we choose $c = -\frac{p_1 p_2 \cdots p_t}{2}$. Then I is the unique integer satisfying $|I| < (p_1 p_2 \cdots p_t)/2$ and satisfying $(*)$.

In the reconstruction of (d,y) from the congruences

$$\left.\begin{array}{l} d \equiv d_k \pmod{p_k} \\ y \equiv y_k \pmod{p_k} \end{array}\right\} \quad 1 \le k \le t$$

$A\overline{y}_k \equiv \overline{d}_k b \pmod{p_k}$ must therefore have been solved for enough primes p_k so that

$$(**) \qquad |d|, \|y\|_\infty < \frac{p_1 p_2 \cdots p_t}{2}$$

$$\text{or} \quad |2d|, \|2y\|_\infty < p_1 p_2 \cdots p_t \quad .$$

To obtain an upper bound on the number of primes required, we note that if $A = (a_{ij})$ is an n by n matrix and $b = (b_i)$ is an n by 1 vector, then y and d are n'th order determinants. If a_{ij} and b_i have precision $\le s$, then (from section 2.2.1) y and d have precision $\le nw$ where:

$$w = \frac{1}{2} \log_m n + \log(c+1) + (s-1)$$

$$m-1 = \text{maximum single-precision integer}$$

$$a_{ij} = \sum_{r=0}^{s-1} c_r^{(ij)} m^r \tag{3.1.3}$$

$$b_j = \sum_{r=0}^{s-1} d_r^{(j)} m^r$$

$$c = \max\left(\max_{1 \le i,j \le n} |c_{s-1}^{(ij)}|, \max_{1 \le j \le n} |d_{s-1}^{(j)}| \right) \quad .$$

Then $2y$ and $2d$ have precision $\leq nw + \log_m 2$ and $t = nw + \log_m 2 + 1$ primes are sufficient to satisfy (**).

An upper bound on the number of modular solutions required is given by t . In some cases t solutions will actually be needed, but in other cases the bound is unnecessarily large. An alternative approach, used by Borosh and Fraenkel [10], is to continue computing terms of the mixed-radix representation of y and d

$$y = y_1 + y_2 p_1 + y_3 p_1 p_2 + \dots + y_k p_1 p_2 \cdots p_{k-1} + \dots$$

$$d = d_1 + d_2 p_1 + d_3 p_1 p_2 + \dots + d_k p_1 p_2 \cdots p_{k-1} + \dots$$

until for some j , $d_j = 0$ and y_j is the zero vector. The probability that the correct solution has been found is high, but a substitution check is made, and additional terms of the mixed-radix representation are found if the check fails.

The substitution check is expensive since it requires operations with multi-precision integers. An alternative termination procedure has been suggested by Cabay [13]. He proved that if $d_k = 0$ and y_k is the zero vector for $m + 1 \leq k \leq m + s$ where $||A||_\infty \leq p_1 p_2 \cdots p_s$ and $||b||_\infty \leq p_1 p_2 \cdots p_s$, and

$$y_I = y_1 + y_2 p_1 + \dots + y_m p_1 p_2 \cdots p_{m-1}$$

$$d_I = d_1 + d_2 p_1 + \dots + d_m p_1 p_2 \cdots p_{m-1}$$

then $Ay_I = d_I b$ and $x = y_I/d_I$ is the solution if $d_I \neq 0$. If $d_I = 0$, $y_I \neq 0$, then A is singular. If $d_I = 0$ and y_I is the zero vector, it is almost certain that A is singular, but to be sure, more primes must be used until the bound (**) is satisfied.

3.1.2 Operation Counts

The assumption is made that if d is a single-precision integer and e is an s-precision integer, calculation of $(d+e)$ requires s additions, $(d\times e)$ requires s multiplications and $(s-1)$ additions, and $(e\div d)$ requires $(2s-2)$ multiplications/divisions and $(s-1)$ additions/subtractions.

The formation of t modular systems, then, requires the following operations:

$$\begin{cases} 0 \;, & s = 1 \\ [t(n^2+n)(2s-2)]M + [t(n^2+n)(s-1)]A \;, & s > 1 \end{cases} \qquad (*)$$

The Gaussian elimination algorithm requires the computation of $a_{hh}^{-1} \pmod{p_k}$ once per iteration on h . Computation of $a^{-1} \pmod{p_k}$ requires an average of $7/12 \log_2 a$ divisions, $7/12 \log_2 a$ multiplications, and $7/12 \log_2 a$ additions/subtractions. Noting that $a_{hh} < m$,

and that division by p_k is performed to keep all integers single-precision, then a single solutions requires:

$$\#\text{multiplications} = \sum_{k=1}^{n-1} (n-1)(n-k+1) + \sum_{k=1}^{n-1} (n-k)$$
$$+ \sum_{k=1}^{n-1} \left(\frac{7}{12}\log_2 m\right) = \frac{1}{3}n^3 + \frac{1}{2}n^2$$
$$+ n\left(\frac{7}{12}\log_2 m - \frac{5}{6}\right) - \frac{7}{12}\log_2 m$$

$$\#\text{additions} = \sum_{k=1}^{n-1} (n-k)(n-k+1) + \sum_{k=1}^{n-1} \left(\frac{7}{12}\log_2 m\right)$$
$$= \frac{1}{3}n^3 + n\left(\frac{7}{12}\log_2 m - \frac{1}{3}\right) - \frac{7}{12}\log_2 m$$

$$\#\text{divisions} = 2\sum_{k=1}^{n-1} (n-k)(n-k+1) + \sum_{k=1}^{n-1} (n-k)$$
$$+ \sum_{k=1}^{n-1} \left(\frac{7}{12}\log_2 m\right) = \frac{2}{3}n^3 + \frac{1}{2}n^2$$
$$+ n\left(\frac{7}{12}\log_2 m - \frac{7}{6}\right) - \frac{7}{12}\log_2 m \quad .$$

During back-substitution, multiplication by the inverse of a_{hh}, $1 \le h \le n$, is performed. Since all of these inverses except a_{nn}^{-1} have been calculated during the triangularization process, it follows that:

$$\#\text{multiplications} = \sum_{k=1}^{n-1} k + n + \frac{7}{12}\log_2 m = \frac{1}{2}n^2 + \frac{1}{2}n$$
$$+ \frac{7}{12}\log_2 m$$

$$\#\text{additions} = \sum_{k=1}^{n-1} k + \frac{7}{12}\log_2 m = \frac{1}{2}n^2 - \frac{1}{2}n + \frac{7}{12}\log_2 m$$

$$\#\text{divisions} = 2\sum_{k=1}^{n-1} k + n + \frac{7}{12}\log_2 m = n^2 + \frac{7}{12}\log_2 m \quad .$$

An additional $2(n-1)$ multiplications/divisions are needed to find $\bar{d}_k = \prod_{i=1}^{n} a_{ii}$ and $2n$ multiplications/divisions are required to find $\bar{y}_k = \bar{x}_k\bar{d}_k$. To find t modular solutions, then, the operations required are:

$$\begin{aligned}
&[t\{(n^3 + \tfrac{5}{2}n^2 + n(\tfrac{7}{6}\log_2 m + \tfrac{5}{2}) - 2\}]M \\
&\quad + [t\{\tfrac{1}{3}n^3 + n(\tfrac{7}{12}\log_2 m - \tfrac{5}{6})\}]A \\
&= [n^3 t + \tfrac{5}{2}n^2 t + nt(\tfrac{7}{6}\log_2 m + \tfrac{5}{2}) - 2t]M \\
&\quad + [\tfrac{1}{3}n^3 t + nt(\tfrac{7}{12}\log_2 m - \tfrac{5}{6})]A \quad . \qquad (**)
\end{aligned}$$

To find the mixed-radix representation (Equation (3.1.1)) of an integer from t modular values requires:

$$\#\text{multiplications} = \sum_{k=1}^{t-2} k + (t-1) = \frac{1}{2}t^2 - \frac{1}{2}t$$

$$\#\text{additions} = \sum_{k=1}^{t-2} k + (t-1) = \frac{1}{2}t^2 - \frac{1}{2}t$$

$$\#\text{divisions} = \#\text{multiplications} + \#\text{additions} = t^2 - t \quad .$$

Application of Horner's rule (Equation (3.1.2)) requires:

$$\#\text{multiplications} = \sum_{k=1}^{t-1} k = \frac{1}{2}t^2 - \frac{1}{2}t$$

$$\#\text{additions} = \sum_{k=1}^{t} k + \sum_{k=1}^{t-1} (k-1) = t^2 - t \quad .$$

To construct the $(n+1)$ integers of d and y therefore requires:

$$\begin{aligned}&[(n+1)(2t^2 - 2t)]M + [(n+1)(\tfrac{3}{2}t^2 - \tfrac{3}{2}t)]A \\ &\quad = [n(2t^2 - 2t) + 2t^2 - 2t]M \\ &\quad + [n(\tfrac{3}{2}t^2 - \tfrac{3}{2}t) + \tfrac{3}{2}t^2 - \tfrac{3}{2}t]A \quad . \qquad (***)\end{aligned}$$

Combining the totals of (*), (**), and (***), the complete solution of $Ay = bd$ requires the following operations:

$$\begin{cases}[n^3t + \frac{5}{2}n^2t + n(\frac{7}{6}t\log_2 m + 2t^2 + \frac{1}{2}t) + 2t^2 - 4t]M, & s=1 \\ [n^3t + n^2(2ts + \frac{1}{2}t) + n(\frac{7}{6}t\log_2 m + 2t^2 - \frac{3}{2}t \\ \qquad + 2ts) + 2t^2 - 4t]M, & s>1\end{cases}$$

$$+\begin{cases}[\frac{1}{3}n^3t + n(\frac{7}{12}t\log_2 m + \frac{3}{2}t^2 - \frac{7}{3}t) + \frac{3}{2}t^2 - \frac{3}{2}t]A, & s=1 \\ [\frac{1}{3}n^3t + n^2(ts-t) + n(\frac{7}{12}t\log_2 m + \frac{3}{2}t^2 - \frac{10}{3}t + ts) \\ \qquad + \frac{3}{2}t^2 - \frac{3}{2}t]A, & s>1\end{cases}$$

Since $\log_m 2 \approx 0$, then $t \approx nw+1$, and the above totals are equivalent to:

$$\begin{cases} [n^4w + n^3(2w^2+\frac{5}{2}w+1) + n^2(2w^2+\frac{7}{6}w\log_2 m+\frac{9}{2}w+\frac{5}{2}) \\ \qquad + n(\frac{7}{6}\log_2 m+\frac{5}{2}) - 2]M , & s=1 \\ [n^4w + n^3(2w^2+2ws+\frac{1}{2}w+1) + n^2(2w^2+\frac{7}{6}w\log_2 m+\frac{5}{2}w+2ws \\ \qquad +2s+\frac{1}{2}) + n(\frac{7}{6}\log_2 m+\frac{1}{2}+2s) - 2]M , & s>1 \end{cases}$$

$$+ \begin{cases} [\frac{1}{3}n^4w + n^3(\frac{3}{2}w^2+\frac{1}{3}) + n^2(\frac{3}{2}w^2+\frac{7}{12}w\log_2 m+\frac{2}{3}w) \\ \qquad + n(\frac{3}{2}w+\frac{7}{12}\log_2 m-\frac{5}{6})]A , & s=1 \\ [\frac{1}{3}n^4w + n^3(\frac{3}{2}w^2+ws-w+\frac{1}{3}) + n^2(\frac{3}{2}w^2+\frac{7}{12}w\log_2 m-\frac{1}{3}w+ws+s-1) \\ \qquad + n(\frac{3}{2}w+\frac{7}{12}\log_2 m - \frac{11}{6}+s)]A , & s>1 \end{cases}$$

where s is the precision of the elements of A and b, and w and m are defined by (3.1.3).

From these totals it can be seen that the congruential method uses $O(n^4w)$ operations. If $s = 1$, $w \leq 1$, then these totals are:

$$[n^4 + \frac{11}{2}n^3 + n^2(9+\frac{7}{6}\log_2 m) + n(\frac{5}{2}+\frac{7}{6}\log_2 m) - 2]M$$
$$+ [\frac{1}{3}n^4 + \frac{11}{6}n^3 + n^2(\frac{13}{2}+\frac{7}{12}\log_2 m) + n(\frac{2}{3}+\frac{7}{12}\log_2 m)]A .$$

3.2 Polynomial Systems

3.2.1 The Method

In this section, the congruential method of solving a system of linear equations $Ax = b$ where the elements of A and b are multi-variate polynomials with

integer coefficients is considered. As in the integer case, d and y satisfying $Ay = db$ are found, and then y/d is the required solution. The components of d and y are polynomials of the form

$$f(x) = \sum_{e_1=0}^{d_1} \cdots \sum_{e_r=0}^{d_r} a_e x_1^{e_1} \cdots x_r^{e_r} ,$$

where $e = (e_1,e_2,\ldots,e_r)$ and $a_e \in I$.

To ensure that none but the initial and final steps require multi-precision operations, the system is solved, as in the integer case, modulo a number of primes p_k . For each prime p_k the solution is represented by $\bar{d}_k$ and $\bar{y}_k$ with coefficients which are congruent modulo p_k to the coefficients of d and y respectively. The final step is construction of the multi-precision coefficients of y and d by means of the Chinese Remainder Theorem for integers. Since the latter process has been described in section 3.1, the problem to be discussed here is that of finding the solution $d \pmod p$, $y \pmod p$ of a system $Ay = db \pmod p$.

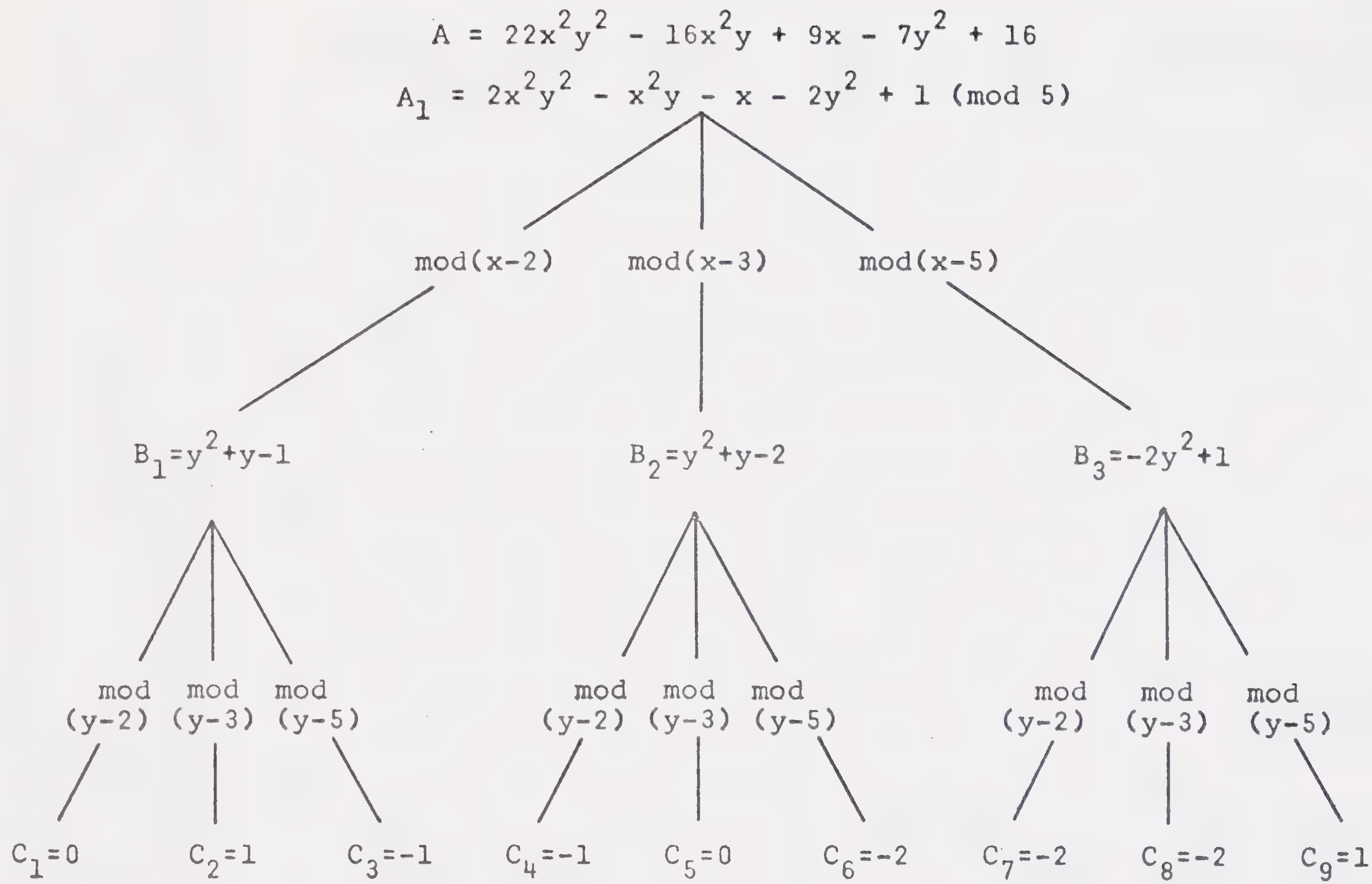
To solve the polynomial system $Ay = db \pmod p$ we first evaluate the polynomials of A and b at an appropriate number of points $c = (c_1,c_2,\ldots,c_r)$. At each point c the system of polynomial equations then becomes an integer system of equations which can be

solved by methods already described. The solution of the integer system represents the polynomial solution of the original system evaluated at the point c . The polynomial solution can be constructed uniquely by interpolation as long as integer solutions have been found for a sufficient number of evaluation points. (See Figure 3.)

In the congruential setting, evaluation of a polynomial $f(x_1,x_2,\ldots,x_r)$ (mod p) at the point c corresponds to finding the residue of $f(x_1,x_2,\ldots,x_r)$ on successive division by p , $(x_1-c_1),\ldots,(x_{r-1}-c_{r-1})$ and (x_r-c_r) . That is, $f(c_1,c_2,\ldots,c_r)$ (mod p) = $f(x_1,x_2,\ldots,x_r)$ mod p , mod$(x_1-c_1),\ldots,$mod(x_r-c_r) where it is understood that mod(x_k-c_k) is performed before mod$(x_{k+1}-c_{k+1})$. In theory, the moduli chosen need not be linear; in practice it is convenient to choose linear moduli of the form (x_i-c_i) so that the residues can be found by evaluation rather than by division.

In the remainder of this section, some of the difficulties which have been encountered with the congruential method are considered.

The first of these occurs when the determinant of an integer system of equations (mod p) at some point $c = (c_1,c_2,\ldots,c_r)$ is zero. This phenomenon will be referred to as a 'bad' prime of type one. It will occur



EVALUATION AND INTERPOLATION MODULO 5

FIGURE 3

when the determinant d of A at the point c is a multiple of p, or when d contains the factor $(x_i - c_i)$ for some i, $i \leq i \leq r$, or when d is actually zero. If d is not zero, McClellan [23] suggests discarding the prime p or the evaluation point, depending on which is responsible for the zero determinant. This discarding process is costly, for several modular solutions previously computed may have to be discarded when p or $(x_i - c_i)$ is found to be 'bad'. This waste is needless in as much as a zero value for $\bar{d}_k$ is perfectly legitimate and $y(c_1, c_2, \ldots, c_r)$ can still be found by Cabay's [13] method.

A second type of 'bad' prime can occur when constructing a polynomial $f(x_1, x_2, \ldots, x_r) \pmod{p}$ from its values at an appropriate number of points c. For polynomials over the integers, the interpolation problem is well-posed as long as f is known at a sufficient number of points. For polynomials over the integers (mod p), the following additional condition is imposed on the i'th component of $(c_1, c_2, \ldots, c_r)$ for all i:

$$(c^{(j)} - c^{(k)}, p) = 1 \quad \text{for every } j,k \text{ such that } j \neq k .$$

A constructive proof that the interpolation problem modulo p is well-posed subject to this condition is given below.

The interpolation process is carried our recursively.

That is, from integers each representing the solution (mod p) at points $(c_1, c_2, \ldots, c_r)$, polynomials in one variable x_r each representing the solution (mod p) evaluated at $(x_1, x_2, \ldots, x_{r-1}) = (c_1, c_2, \ldots, c_{r-1})$ are constructed; then polynomials in two variables x_r and x_{r-1} each representing the solution (mod p) evaluated at $(x_1, x_2, \ldots, x_{r-2}) = (c_1, c_2, \ldots, c_{r-2})$ are found, and so on, until a solution in r variables is obtained. This process of constructing a polynomial (mod p) from its modular values is illustrated in Figure 3. Therefore it is sufficient to show that a polynomial $f(x_1, x_2, \ldots, x_m)$ (mod p) of degree d_m in the variable x_m can be constructed uniquely from $d_m + 1$ polynomials, u_i for $1 \le i \le d_m + 1$, in the variables $x_1, x_2, \ldots, x_{m-1}$ satisfying

$$f(x_1, x_2, \ldots, x_{m-1}, c_m^{(i)}) \equiv u_i(x_1, x_2, \ldots, x_{m-1}) \pmod p\ ,$$
$$1 \le i \le d_m + 1\ .$$

<u>Theorem</u>: Let p be an odd prime and let $c_m^{(1)}, c_m^{(2)}, \ldots c_m^{(d_m+1)}$ be distinct integers modulo p. Let $u_1, u_2, \ldots, u_{d_{m+1}}$ be polynomials in the variables $x_1, x_2, \ldots, x_{m+1}$. Then there exists a unique polynomial

$$f(x_1, x_2, \ldots, x_m) = \sum_{e_1=0}^{d_1} \cdots \sum_{e_m=0}^{d_m} a_e x_1^{e_1} \cdots x_m^{e_m}$$

such that

$$f(x_1,x_2,\ldots,x_{m-1},c_m^{(i)}) \equiv u_i \pmod p$$

for $1 \le i \le d_m+1$ and $|a_e| \le \frac{p-1}{2}$ for all e .

<u>Proof</u>: (i) Existence of f .

Let $a_1 = s_1 = u_1$

$$a_k = (u_k - s_{k-1}) \times \prod_{i=1}^{k-1} (x_m - c_m^{(i)})^{-1}$$

$$[\mathrm{mod}\ p, \mathrm{mod}(x_m - c_m^{(k)})]$$

$$s_k = s_{k-1} + a_k \cdot \prod_{i=1}^{k-1} (x_m - c_m^{(i)}) [\mathrm{mod}\ p] \quad .$$

The proof that $s_k \equiv u_i\ [\mathrm{mod}\ p, \mathrm{mod}(x_m - c_m^{(i)})]$,

$1 \le i \le k$ for $k=1,2,\ldots,d_m+1$ is by induction.

For $k=1$, $s_k = s_1 = u_1 [\mathrm{mod}\ p, \mathrm{mod}(x_m - c_m^{(1)})]$ by definition.

Assume $s_{k-1} \equiv u_i [\mathrm{mod}\ p, \mathrm{mod}(x_m - c_m^{(i)})]$, $1 \le i \le k-1$.

Then $s_k = s_{k-1} + a_k \prod_{i=1}^{k-1} (x_m - c_m^{(i)})\ [\mathrm{mod}\ p]$

$$= s_{k-1} + \{(u_k - s_{k-1}) \times \prod_{k=1}^{k-1} (x_m - c_m^{(i)})^{-1}$$

$$[\mathrm{mod}\ p, \mathrm{mod}(x_m - c_m^{(k)})]\} \times \prod_{i=1}^{k-1} (x_m - c_m^{(i)})\ [\mathrm{mod} p].$$

Trivially $s_k \equiv u_i\ [\mathrm{mod}\ p, \mathrm{mod}(x_m - c_m^{(i)})]$,

$1 \le i \le k-1$.

To show that $s_k \equiv u_k\ [\mathrm{mod}\ p, \mathrm{mod}(x_m - c_m^{(k)})]$

observe that

$$(x_m - c_m^{(i)})^{-1} \text{ [mod } p, \text{mod}(x_m - c_m^{(k)})] =$$

$$\frac{1}{c_m^{(k)} - c_m^{(i)}} \text{ [mod } p] \quad \text{is well-defined since}$$

$$(c_m^{(k)} - c_m^{(i)}, p) = 1 \quad .$$

Thus $s_k \equiv s_{k-1} + (u_k - s_{k-1}) \times \prod_{i=1}^{k-1} (x_m - c_m^{(i)})^{-1}$

$$(x_m - c_m^{(i)}) \quad \text{[mod } p, \text{mod}(x_m - c_m^{(k)})]$$

$$= s_{k-1} + (u_k - s_{k-1}) \times 1 \text{ [mod } p, \text{mod}(x_m - c_m^{(k)})]$$

$$= s_{k-1} + u_k - s_{k-1} \text{ [mod } p, \text{mod}(x_m - c_m^{(k)})]$$

$$= u_k \text{ [mod } p, \text{mod}(x_m - c_m^{(k)})] \quad .$$

$\therefore\ s_{d_m+1} \equiv u_i \quad [\text{mod } p, \text{mod}(x_m - c_m^{(i)})] \ , \quad 1 \le i \le d_m + 1$.

Also, by definition, the absolute value of the coefficients of $s_{d_m+1} \le \frac{1}{2}(p-1)$.

(ii) Uniqueness of f .

Suppose there exists a polynomial $q(x_1, \ldots, x_m)$

$$= \sum_{e_1=0}^{d_1} \cdots \sum_{e_m=0}^{d_m} b_e x_1^{e_1} \ldots x_m^{e_m} \ , |b_e| \le \tfrac{1}{2}(p-1) \ ,$$

such that $q \equiv u_i$ [mod p, mod$(x_m - c_m^{(i)})$] ,

$1 \le i \le d_m + 1$.

Then $q \equiv s_{d_m+1}$ [mod p, mod$(x_m - c_m^{(i)})$] , $1 \le i \le d_m + 1$,

and $q - s_{d_m+1} \equiv c \prod_{i=1}^{d+1} (x_m - c_m^{(i)})$ [mod p] .

But degree of q in x_m = degree of s_{d_m+1} in $x_m = d_m$.

Thus $c = 0$ [mod p]

and $q = s_{d_m+1}$ [mod p] .

Then since $|a_e| \leq \frac{1}{2}(p-1)$, $|b_e| \leq \frac{1}{2}(p-1)$, we have $q = s_{d_m+1}$.

s_{d_m+1} is therefore the required unique polynomial.

The above theorem implies that if $d \pmod p$ is a polynomial of degree d_i in r variables x_i , $1 \leq i \leq r$, then $d \pmod p$ can be constructed uniquely from its values at $\prod_{i=1}^{r}(d_i+1)$ points $c = (c_1, c_2, \ldots, c_r)$ providing that p is chosen so that $(c_m^{(i)} - c_m^{(k)}, p) = 1$ if $i \neq k$. Since p is chosen as a very large single-precision prime, and it is convenient to choose small evaluation points, this restriction is easily satisfied, and 'bad' primes of type two will not occur.

If $r_i^{(k)} = c^{(k)} - c^{(i)}$ and $b_k = \prod_{i=1}^{k-1}(c^{(k)} - c^{(i)})$, $2 \leq k \leq d+1$, are pre-computed, the Newtonian interpolation algorithm for constructing a polynomial $f(x)$ of degree d from the values u_i where $f(x) \equiv u_i$ [mod p, mod $(x-c^{(k)})$] is:

$a_1 = u_1$;

For $k = 2$ to $(d+1)$ do (3.2.1)

$$v = a_{k-1} ;$$

$$\text{For } i = k-2 \text{ to } 1 \text{ do} \qquad (3.2.1)$$

$$v = (v.r_i^{(k)} + a_i) \ [\text{mod } p] ;$$

$$a_k = (u_k - v) \times b_k^{-1} \ [\text{mod } p] \quad .$$

Then by Horner's rule

$$f = (\ldots(a_{d+1}(x-c^{(d)})+a_d)(x-c^{(d-1)})+a_{d-1})\ldots+a_2)(x-c^{(1)}) + a_1 \quad .$$

If the elements of A and b are polynomials of degree d in each of r variables, having single-precision coefficients,then (from section 2.3.1) d and y are polynomials of degree $\leq nd$ in each of r variables and the coefficients of d and y have precision $\leq nw$ where

$$w = \frac{1}{2} \log_m n + \log_m c$$

$m-1$ = maximum single-precision integer

c = approximate value of coefficients of polynomials in A and b .

It is therefore sufficient to find solutions at $(nd+1)^r$ points modulo t prime integers, where

$$t = nw + \log_m 2 + 1 \quad . \qquad (3.2.2)$$

In some cases, fewer than $(nd+1)^r$ solutions are

required to find a solution modulo p . Again one approach (used by McClellan [23]) is to cease computing terms of the mixed-radix representation when the same solution is found for two successive moduli and to do a substitution check. The following theorem shows that Cabay's [13] termination algorithm can be extended to polynomial interpolation.

<u>Theorem</u>: Let $A = (a_{ij})$ be an n by n matrix where a_{ij} is a polynomial in variables which include x and $\max_{1 \leq j \leq n} [\deg_x (a_{ij})] \leq r_j$. Let $b = (b_i)$ be an n by 1 vector where b_i is a polynomial in variables which include x and $\max_{1 \leq i \leq n} [\deg_x (b_i)] \leq t_i$.

If the mixed-radix representation of $y \pmod p$, $d \pmod p$, $p \in I$, is

$$y = y_I + (x-c_1)(x-c_2)\ldots(x-c_m).0 + \ldots + (x-c_1)(x-c_2)\ldots(x-c_{m+s-1}).0 + (x-c_1)\ldots(x-c_{m+s}).y_R$$

$$d = d_I + (x-c_1)(x-c_2)\ldots(x-c_m).0 + + (x-c_1)(x-c_2)\ldots(x-c_{m+s-1}).0 + (x-c_1)\ldots(x-c_{m+s}).d_R$$

where

$$y_I = y_1 + (x-c_1)y_2 + \dots$$
$$+ (x-c_1)(x-c_2)\dots(x-c_{m-1})y_m$$

$$d_I = d_1 + (x-c_1)d_2 + \dots$$
$$+ (x-c_1)(x-c_2)\dots(x-c_{m-1})d_m$$

and $\max_x [\deg (d_I)] \le q$

$\max_x [\deg (y_I)_i] \le p_i \ , \quad 1 \le i \le n$

$\max [\max_{1 \le i \le n} (r_i + p_i), \max_{1 \le i \le n} (t_i + q)] \le m+s-1$

then $Ay_I \equiv d_I b \pmod p$.

Proof: Assume the contrary. That is, suppose $Ay_I - d_I b \not\equiv 0 \pmod p$. Then there exists a polynomial $c \not\equiv 0 \pmod p$ such that

$$0 \equiv (Ay-db)_i \equiv (Ay_I - d_I b)_i$$
$$+ c(x-c_1)(x-c_2)\dots(x-c_{m+s}) \pmod p \ .$$

Then $[\deg_x (Ay_I - d_I b)_i] \ge m+s$.

But $\deg_x (Ay_I - d_I b)_i \le \max [\deg_x (Ay_I)_i, \deg_x (d_I b)_i]$

$\le \max [\max_{1 \le i \le n} (r_i + p_i), \max_{1 \le i \le n} (t_i + q)]$

$\le m+s-1$.

Contradiction.

$Ay_I = d_I b \pmod p$. Q.E.D.

If $d_I \neq 0$ then $x = y_I/d_I \pmod p$. If y_I is not the zero vector and $d_I = 0$, then $A \pmod p$ is singular. If d_I and y_I are both zero, then it is almost certain that $A \pmod p$ is singular, but to be sure the $(nd+1)$ modular solutions indicated by Hadamard's bound must be found.

Note, however, that $Ay_I = d_I b \pmod p$ does not imply that $d \equiv d_I \pmod p$, $y \equiv y_I \pmod p$. We know only that $y_i/d_I = y/d \pmod p$. Then either $d \equiv d_I \pmod p$, $y \equiv y_I \pmod p$, or d and y have a common factor (mod p) which is not present in d_I and y_I . The early termination which occurs in the latter case is an advantage when integer systems are being solved, since it means that a solution for x has been found without solving completely for d and y .

In the polynomial case, however, early termination means that interpolation on d_I and y_I to find d and y may produce an incorrect result. In Figure 4, for example, termination modulo 3 occurs with $d_I = 1$, while modulo 5 the partial solution obtained is $d_I = 1 - 2.x + x(x-1) = x^2 + 2x + 1 \pmod 5$. Since $d = x^2 + 2x + 1 \pmod 5$, but $d \neq 1 \pmod 3$ (where d

$$[A,b] = \begin{bmatrix} x & -1 & 2x \\ 1 & x+2 & 2 \end{bmatrix}$$

mod 3

$$\begin{bmatrix} x & -1 & -x \\ 1 & x-1 & -1 \end{bmatrix}$$

$x=0$ $\qquad$ $x=1$ $\qquad$ $x=2$

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & -1 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$\bar{d} = 1+0.x$$

$$\bar{y} = \begin{bmatrix} 2+0.x \\ 0+0.x \end{bmatrix}$$

$$\bar{d} = 1+0.x+x(x-1)$$

$$\bar{y} = \begin{bmatrix} 2+0.x+2x(x-1) \\ 0+0.x+0.x(x-1) \end{bmatrix}$$

mod 5

$$\begin{bmatrix} x & -1 & 2x \\ 1 & x+2 & 2 \end{bmatrix}$$

$x=0$ $\qquad$ $x=1$ $\qquad$ $x=2$

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 2 & 2 \end{bmatrix} \quad \begin{bmatrix} 1 & -1 & 2 \\ 1 & 3 & 2 \end{bmatrix} \quad \begin{bmatrix} 2 & -1 & -1 \\ 1 & -1 & 2 \end{bmatrix}$$

$$\bar{d} = 1-2.x+x(x-1)+0.x(x-1)(x-2)$$

$$\bar{y} = \begin{bmatrix} 2+x+2.x(x-1)+0.x(x-1)(x-2) \\ 0+0.x+0.x(x-1)+0.x(x-1)(x-2) \end{bmatrix}$$

$$d = 1+3x+x(x-1)=x^2+2x+1$$

$$y = \begin{bmatrix} 2+6x+2x(x-1)=2x^2+4x2 \\ 0+0.x+0.x(x-1)=0 \end{bmatrix}$$

TERMINATION OF POLYNOMIAL INTERPOLATION

FIGURE 4

is the true solution), it is clear that the two d_I's obtained are not modular representations of the same polynomial, and interpolation using these two partial solutions will not yield the correct solution. This case will be referred to as 'bad' primes of type three. In an algorithm, the fact that 3 is a 'bad' prime would be recognized by observing that three non-zero terms of the mixed-radix representation (mod 5) were found. Therefore three terms of the modulo 3 solution are required to ensure that modular representations of the same polynomial solution are obtained in each case. Since only two terms were computed (mod 3), the modulo 3 partial solution may be incorrect.

Interpolation (and the termination algorithm) are applied recursively, starting with the integer solutions obtained and working upwards in the tree to obtain first solutions in one variable, then solutions in two variables, and so on. Therefore 'bad' primes may occur at any level in the tree. That is, a 'bad' prime may be an integer modulus p_k (as in Figure 4) or a polynomial modulus $(x_i-c^{(k)})$.

To ensure that early termination does not cause an incorrect result, McClellan [23] discards the modulus $(x_i-c^{(k)})$ (or the prime p_k , as the case may be) if $\bar{d}_k$ or $\bar{y}_k$ have lower degree than $\bar{d}_j$ and $\bar{y}_j$ for some other modulus $(x_i-c^{(j)})$ (or p_j). However, it has

been noted that the terms of the mixed-radix representation of $\bar{d}_k$ and $\bar{y}_k$ which have been obtained are not incorrect, but that more of them are required. Instead of discarding the modulus, the additional terms can be computed. In Figure 4, for example, the additional term required modulo 3 (shown in dotted lines) can be computed to give the correct result.

Two observations can be made. First, since s zero coefficients of the mixed-radix representations of d_I and y_I are computed before terminating modulo $(x_i-c^{(k)})$ (or mod p_k), more terms are needed only if the degree of some other solution d_I, y_I modulo $(x_i-c^{(j)})$ (or mod p_j) is greater by at least $(s+1)$. Second, if the 'bad' prime occurs after any 'good' prime, computation of the additional terms is not troublesome because it is known at the time of computation that extra terms are required.

In the event that a 'bad' prime occurs before a 'good' prime, additional terms of the mixed-radix representation must be added to a previous solution. To do so, the previous solution must be known. In general, solutions obtained lower in the tree are released from storage as solutions in more variables are obtained. However, it is always possible to regain solutions lower in the tree by evaluating a solution higher in the tree. For example, if a polynomial partial solution $\bar{d}(x_1,x_2)$

is known, a previous solution $\overline{d}(x_1)$ such that $\overline{d}(x_1 . x_2) \equiv \overline{d}(x_1)$ [mod $(x_1 - c)$] can be obtained by evaluating $\overline{d}(x_1, x_2)$ at $x = c$. The computation required to do so is certainly less than that required to solve again, or to solve with another prime.

In some cases, no previous solutions have to be found before an additional term can be added. In Figure 4, for example, $d_I \equiv 1 \pmod 3$, $y_I \equiv (2,0) \pmod 3$ are immediately available when the modulo 5 solution indicates that an additional term is required. The additional term can be computed without obtaining again the solutions at $x = 0$, $x = 1 \pmod 3$.

'Bad' primes of type three, then, can always be used, and can often be used with no loss of computation. Additional computation is sometimes required, but it is certainly less than that required when the prime is discarded.

3.2.2 Operation Counts

As in Chapter Two, the analysis is restricted to the case of polynomials of degree d in each of r variables and having single-precision coefficients. It is therefore sufficient to find solutions at $(nd+1)^r$ points modulo t primes p_k where t is defined by (3.2.1).

If evaluation is performed by Horner's rule, and division is performed to keep all integers modulo p_k, then:

$$\begin{aligned}
\#\text{multiplications} &= t(n^2+n)\sum_{k=1}^{r}[(d+1)^{r-k}(d)(nd+1)^k] \\
&\le t(n^2+n)\sum_{k=1}^{r}[(d+1)^{r-k}(d)(n^k)(d+1)^k] \\
&= t(n^2+n)(d)(d+1)^r\sum_{k=1}^{r}n^k \\
&\approx td(n^2+n)(d+1)^r\,n^r \\
&= (d+1)^r(tdn^{r+2}+tdn^{r+1})
\end{aligned}$$

$$\#\text{additions} = \#\text{multiplications} = (d+1)^r(tdn^{r+2}+tdn^{r+1})$$

$$\begin{aligned}
\#\text{divisions} &= \#\text{multiplications} + \#\text{additions} \\
&= (d+1)^r(2tdn^{r+2}+2tdn^{r+1}) \quad .
\end{aligned}$$

From section 3.1.2, the operations required to solve $t(nd+1)^r$ integer systems modulo p_k are:

$$\begin{aligned}
\#\text{multiplications/divisions} = t(nd+1)^r\{n^3 + \tfrac{5}{2}n^2 \\
+ n(\tfrac{7}{6}\log_2 m + \tfrac{5}{2}) - 2\}
\end{aligned}$$

$$\begin{aligned}
\#\text{additions/subtractions} = t(nd+1)^r\{\tfrac{1}{3}n^3 \\
+ n(\tfrac{7}{6}\log_2 m - \tfrac{5}{6})\} \quad .
\end{aligned}$$

When finding the mixed-radix representation of $d \pmod{p_k}$, $y \pmod{p_k}$, $1 \le k \le t$, by algorithm (3.2.1),

at the j'th step $r_i^{(k)}$ is an integer, v, a_i, and u_k are polynomials of degree nd in each of $(j-1)$ variables, and interpolation is performed $(nd+1)^{r-j}$ times. Then the operations required are:

$$\begin{aligned}
\#\text{multiplications} &= t(n+1) \sum_{j=1}^{r} \{ \sum_{k=1}^{nd} [k(nd+1)^{j-1} \\
&\qquad + \frac{7}{12} nd \log_2 m] \times (nd+1)^{r-j}\} \\
&= t(n+1) \sum_{j=1}^{r} \{[(nd+1)^j(\frac{1}{2} nd) + \frac{7}{12} nd \log_2 m] \\
&\qquad \times (nd+1)^{r-j}\} \\
&= t(n+1) \sum_{j=1}^{r} \{\frac{1}{2} nd(nd+1)^r + \frac{7}{12} nd \log_2 m(nd+1)^{r-j}\} \\
&= t(n+1)\{\frac{1}{2} ndr(nd+1)^r + \frac{7}{12} \log_2 m(nd+1)^{r+1} \\
&\qquad - \frac{7}{12} nd \log_2 m - 2 \log_2 m\} \\
&= (nd+1)^r[n^2(\frac{1}{2} drt + \frac{7}{12} td \log_2 m) + n(\frac{7}{12} td \log_2 m \\
&\qquad + \frac{7}{12} t \log_2 m + \frac{1}{2} drt) + \frac{7}{12} t \log_2 m] \\
&\qquad - \frac{7}{12} tdn^2 \log_2 m - \frac{7}{12} tdn \log_2 m \\
&\qquad - \frac{7}{12} tn \log_2 m - \frac{7}{12} t \log_2 m
\end{aligned}$$

$$\#\text{additions} = \#\text{multiplications}$$

$$\begin{aligned}
\#\text{divisions} &= t(n+1) \sum_{j=1}^{r} \{2 \sum_{k=1}^{nd-1} k(nd+1)^{j-1} \\
&\qquad + 2nd(nd+1)^{j-1} + \frac{7}{12} nd \log_2 m] \\
&\qquad (nd+1)^{r-j}\}
\end{aligned}$$

$$= (nd+1)^r[n^2(drt + \frac{7}{12} td \log_2 m)$$

$$+ n(\frac{7}{12} td \log_2 m + \frac{7}{12} t \log_2 m + drt)$$

$$+ \frac{7}{12} t \log_2 m] - \frac{7}{12} tdn^2 \log_2 m - \frac{7}{12} tdn \log_2 m$$

$$- \frac{7}{12} tn \log_2 m - \frac{7}{12} t \log_2 m \quad .$$

To find $d \pmod{p_k}$, $y \pmod{p_k}$ from the mixed-radix form by Horner's rule requires:

$$\#\text{multiplications} = t(n+1) \sum_{j=1}^{r} [\frac{1}{2} nd(nd+1)^j(nd+1)^{r-j}]$$

$$= t(n+1)(\frac{1}{2} drn)(nd+1)^r$$

$$= (nd+1)^r(\frac{1}{2} tdrn^2 + \frac{1}{2} tdrn)$$

$$\#\text{additions} = t(n+1) \sum_{j=1}^{r} [nd(nd+1)^j(nd+1)^{r-j}]$$

$$= (nd+1)^r (tdrn^2+tdrn)$$

$$\#\text{divisions} = \#\text{multiplications} + \#\text{additions}$$

$$= (nd+1)^r(\frac{3}{2} tdrn^2 + \frac{3}{2} tdrn) \quad .$$

Construction of the multi-precision coefficients of d and y from $d \pmod{p_k}$, $y \pmod{p_k}$, $1 \le k \le t$, requires (from section 3.1.2):

$$\#\text{multiplications/divisions} = (nd+1)^r(n+1)(2t^2-2t)$$

$$= (nd+1)^r(2t^2n-2tn+2t^2-2t)$$

$$\#\text{additions} = (nd+1)^r(n+1)(\tfrac{3}{2}\,t^2 - \tfrac{3}{2}\,t)$$
$$= (nd+1)^r\ (\tfrac{3}{2}\,t^2n - \tfrac{3}{2}\,tn + \tfrac{3}{2}\,t^2 - \tfrac{3}{2}\,t)$$

Therefore the complete solution requires the following arithmetic operations:

#multiplications/divisions =

$$= (nd+1)^r[n^3t + n^2(\tfrac{7}{2}\,drt + \tfrac{5}{2}\,t + \tfrac{7}{6}\,td\,\log_2 m)$$
$$+ \tfrac{7}{6}\,t\,\log_2 m] + (d+1)^r(3tdn^{r+2} + 2tdn^{r+1})$$
$$- \tfrac{7}{6}\,t\,\log_2 m(dn^2 + dn + n - 1)$$

#additions/subtractions

$$= (nd+1)^r[\tfrac{1}{3}\,n^3t + n^2(2drt + \tfrac{7}{12}\,td\,\log_2 m)$$
$$+ n(2drt + \tfrac{3}{2}\,t^2 - \tfrac{7}{3}\,t + \tfrac{7}{12}\,td\,\log_2 m + \tfrac{7}{12}\,t\,\log_2 m)$$
$$+ \tfrac{3}{2}\,t^2 - \tfrac{3}{2}\,t + \tfrac{7}{12}\,t\,\log_2 m] + (d+1)^r(tdn^{r+2}+tdn^{r+1})$$
$$- \tfrac{7}{12}\,t\,\log_2 m(dn^2 + dn + n - 1)$$

Since $\log_m 2 \simeq 0$ then $t \simeq nw + 1$, and the totals are:

#multiplications/divisions

$$= (nd+1)^r[n^4w + n^3(2w^2 + \tfrac{7}{2}\,drw + \tfrac{5}{2}\,w + \tfrac{7}{6}\,wd\,\log_2 m+1)$$
$$+ n^2(2w^2 + \tfrac{7}{2}\,drw + \tfrac{5}{2}\,w + \tfrac{7}{6}\,wd\,\log_2 m + \tfrac{7}{2}\,dr$$
$$+ \tfrac{7}{6}d\,\log_2 m + \tfrac{5}{2}) + n(2w + \tfrac{7}{3}\,w\,\log_2 m + \tfrac{7}{6}\,d\,\log_2\,m$$
$$+ \tfrac{7}{2}\,dr + \tfrac{5}{2}) + \tfrac{7}{3}\,\log_2 m]$$

$$+ (d+1)^r(3dwn^{r+3} + 3dn^{r+2} + 2dwn^{r+2} + 2dn^{r+1})$$

$$- \frac{7}{6} nw \log_2 m(dn^2+dn+n-1) - \frac{7}{6} \log_2 m(dn^2+dn+n-1)$$

#additions/subtractions

$$= (nd+1)^r[\frac{1}{3} n^4 w + n^3(\frac{3}{2} w^2 + 2drw + \frac{7}{12} dw \log_2 m)$$

$$+ n^2(\frac{3}{2} w^2 + 2drw + \frac{2}{3} w + \frac{7}{12} wd \log_2 m + 2dr$$

$$+ \frac{7}{12} d \log_2 m + \frac{1}{3}) + n(\frac{3}{2} w + \frac{7}{6} w \log_2 m + \frac{7}{12} d \log_2 m$$

$$+ 2dr - \frac{5}{6}) + \frac{7}{6} \log_2 m] + (d+1)^r(dwn^{r+3} + dn^{r+2}$$

$$+ dwn^{r+2} + dn^{r+1}) - \frac{7}{12} nw \log_2 m(dn^2 + dn + n - 1)$$

$$- \frac{7}{12} \log_2 m(dn^2 + dn + n - 1) \quad .$$

The number of multiplications/divisions required is approximately $wn^{r+4}d^r$ and the number of additions/subtractions is approximately $\frac{1}{3} wn^{r+4}d^r$. For univariate polynomials these totals are wn^4d and $\frac{1}{3} wn^4d$ respectively.

If the r variables x_i , $1 \le i \le r$, have varying degrees d_i , then the above totals become $wn^{r+4}(\prod_{i=1}^{r} d_i)$ and $\frac{1}{3} wn^{r+4}(\prod_{i=1}^{r} d_i)$.

CHAPTER FOUR

Conclusions

In this chapter the multi-step and congruential methods are compared on the basis of the results established in Chapters Two and Three. The order of complexity of the two methods are noted, and some observations are made about when one method can be expected to require fewer arithmetic operations than the other.

From Chapter Two, the complexity of the one-step, two-step, (and in general multi-step) methods is $O(n^5w^2)$ for systems of n equations with integer coefficients and $O(n^{2r+r}(\prod_{i=1}^{r} d_i)w^2)$ for systems with polynomial coefficients of degree d_i in the variables x_i , $1\leq i\leq r$, where w is defined by (2.2.1) and (2.3.1) respectively. Even with fast multiplication, these orders are at least $n^4w \log n$ and $n^{r+4}(\prod_{i=1}^{r} d_i)w \log n$. For the congruential method, it was established in Chapter Three that these orders are n^4w and $n^{r+4}(\prod_{i=1}^{r} d_i)w$ respectively. The congruential method is therefore certainly superior for large n . It is stressed that this conclusion contradicts that of Bareiss [3,4], who states that with optimal fast arithmetic the multi-step methods have the same order of complexity as the congruential

methods, but with a smaller leading coefficient.

We consider next the values of n for which the congruential method for integers uses fewer operations. Assume that the linear equations have single-precision coefficients such that

$$\frac{1}{2} \log_m n + \log_m (c+1) \leq 1 \tag{4.1}$$

where $(m-1)$ is the largest single-precision integer and c is defined as in (3.1.3). (That is, for small n, the coefficients are slightly less than the maximum single-precision value.) Then for the two-step method with classical multiplication, the operations required are:

$$[\frac{1}{15} n^5 + \frac{2}{3} n^4 + \frac{19}{12} n^3 + \frac{19}{12} n^2 + \frac{28}{5} n + 2]M$$

$$+ [\frac{3}{20} n^5 + \frac{193}{24} n^2 + \frac{47}{12} n^3 - \frac{53}{12} n^2 - \frac{8}{5} n - 2]A \quad .$$

For the congruential method, the total is:

$$[n^4 + \frac{11}{2} n^3 + n^2(9 + \frac{7}{6} \log_2 m) + n(\frac{5}{2} + \frac{7}{6} \log_2 m) - 2]M$$

$$+ [\frac{1}{3} n^4 + \frac{11}{6} n^3 + n^2(\frac{13}{2} + \frac{7}{12} \log_2 m) + n(\frac{2}{3} + \frac{7}{12} \log_2 m)]A.$$

The number of operations required by the congruential method depends on the word-size of the computer being used. For a 32-bit word (where one bit is a sign bit)

$$\log_2 m = \log_2 (2^{31}) = 31 \quad .$$

Then for the congruential method the arithmetic operations required are:

$$[n^4 + \frac{11}{2} n^3 + \frac{271}{6} n^2 + \frac{116}{3} n - 2]M$$

$$+ [\frac{1}{3} n^4 + \frac{11}{6} n^3 + \frac{81}{4} n^2 + \frac{225}{12} n]A \quad .$$

In this case, the congruential method uses fewer multiplications/divisions for $n > 16$, fewer additions/subtractions for $n \geq 2$, and fewer operations in total for $n \geq 5$.

In the polynomial case, a reliable exact comparison cannot be made on the basis of the approximate operation counts obtained. However, since the congruential method is asymtotically better than the multi-step methods by a factor of $n^{r+1}(\prod_{i=1}^{r} d_i)w$, the method may be superior for smaller n than in the integer case if r and d_i are large. Experimental evidence is needed here, (including experiments with algorithms using fast multiplication techniques) to establish the cross-over point at which the congruential method is superior. McClellan [23] is currently working on such experiments.

It should be noted that as the word size of the computer decreases, the performance of the congruential method in relation to the multi-step methods improves. With a word-size of 16 bits, for example, the congruential method uses fewer total operations for $n \geq 4$ (for the

integer case when equation (4.1) is satisfied).

In conclusion, two observations not central to the theme of this study, but never-the-less of interest, can be made.

First, the congruential algorithm described in Chapter Three is not optimal in a theoretical sense, since an $O(n^{3.8}w)$ method for solving systems of linear equations exists. Strassen [26] has devised fast matrix multiplication and inversion algorithms which are $O(n^{2.8})$. In the congruential setting, this yields an $O(n^{3.8}w)$ method for integers. However, the coefficient preceding $n^{3.8}w$ is very large (approximately 12) . Since $12\ n^{3.8} < n^4$ only when $n > 12^5$, the method, while theoretically of interest, is not practical.

Second, if multiplication and division are more costly operations than addition and subtraction, as they are for many computers, a method proposed by Winograd [28] is of interest. He uses block Gaussian elimination in conjunction with matrix multiplication and inversion algorithms which exchange about half of the multiplication/division operations for additions/subtractions. The total operation count is approximately the same as traditional Gaussian elimination, but fewer of these are multiplications/divisions. This method has the disadvantage (as does Strassen's) that certain submatrices of the coefficient matrix must be non-singular.

REFERENCES

1. E.H. Bareiss, "Multistep Integer-Preserving Gaussian Elimination", Argonne National Lab. Rep. No. ANL-7213 (1966).

2. E.H. Bareiss, "Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination", Math. Comp. 22 (1968), 565-578.

3. E.H. Bareiss, "Computational Solutions of Matrix Problems over an Integral Domain", Argonne National Technical Memorandum No. 222 (1970).

4. E.H. Bareiss, "Computational Solutions of Matrix Problems over an Integral Domain", J. Inst. Maths. Applics. 10 (1972), 68-104.

5. E.H. Bareiss and D. Mazukelli, "Fast Algorithms and Fast Arithmetic in Linear Algebra", Argonne National Lab. Rep. No. ANL-7897 (1971).

6. W.A. Blankinship, "A New Version of the Euclidean Algorithms", Amer. Math. Month. 70 (1963), 742-745.

7. W.A. Blankinship, "Algorithm 288: Solution of Simultaneous Linear Diophantine Equations", Comm. ACM 9 (July 1966), 514.

8. E. Bodewig, Matrix Calculus, North-Holland, 1959.

9. A.B. Borodin and I. Munro, Analysis of Algorithms Lecture Notes, Preliminary Draft, June 1972.

10. I. Borosh and A.S. Fraenkel, "Exact Solutions of Linear Equations with Rational Coefficients by Congruence Techniques", Math. Comp. 20 (1966), 107-112.

11. G.H. Bradley, "Algorithm and Bound for the Greatest Common Divisor of n Integers", Comm. ACM 13 (1970), 433-436.

12. G.H. Bradley, "Algorithms for Hermite and Smith Normal Matrices and Linear Diophantine Equations", Math. Comp. 25 (1971), 897-907.

13. S. Cabay, "Exact Solution of Linear Equations", Proc. 2nd Symp. Symbolic Algebraic Manipulation, Los Angeles, 1971.

14. G.E. Collins, "Computing Multiplicative Inverses in GF(p)", Math. Comp. 23 (1969), 197-200.

15. L. Fox, An Introduction to Numerical Linear Algebra, Clarendon Press, 1964.

16. J.A. Howell and R.T. Gregory, "An Algorithm for Solving Linear Algebraic Equations using Residue Arithmetic I", BIT 9 (1969), 200-224.

17. J.A. Howell and R.T. Gregory, "An Algorithm for Solving Linear Algebraic Equations using Residue Arithmetic II", BIT 9 (1969), 324-337.

18. J.A. Howell and R.T. Gregory, "Solving Linear Equations using Residue Arithmetic - Algorithm II", BIT 10 (1970), 23-37.

19. D.E. Knuth, The Art of Computer Programming Vol. 2: Seminumerical Algorithms, Addison-Wesley, 1969.

20. J.D. Lipson, "Symbolic Methods for the Computer Solution of Linear Equations with Applications to Flowgraphs", Proc. 1968 Summer Institute on Symbolic Mathematical Computation, IBM Programming Lab. Rep. FSC69-0312, June 1969.

21. J.D. Lipson, "Chinese Remainder and Interpolation Algorithms", Proc. 2nd Symp. Symbolic Algebraic Manipulation, Los Angeles, 1971

22. H.A. Luther and L.F. Guseman, Jr., "A Finite Sequentially Compact Process for the Adjoints of Matrices over Arbitrary Integral Domains", Comm. ACM 5 (1962), 447-448.

23. M.T. McClellan, "The Exact Solution of Systems of Linear Equations with Polynomial Coefficients", Proc. 2nd Symp. Symbolic Algebraic Manipulation, Los Angeles, 1971.

24. M. Newman, "Solving Equations Exactly", J. Res. Nat. Bureau Standards-B 71B (1967), 171-179.

25. J.B. Rosser, "A Method of Computing Exact Inverses of Matrices with Integer Coefficients", J. Res. Nat. Bureau Standards, 49 (1952), 349-358.

26. V. Strassen, "Gaussian Elimination is not Optimal", Num. Math. 13 (1969), 354-356.

27. H. Takahasi and Y. Ishibashi, "A New Method for 'Exact Calculation' by a Digital Computer", Information Processing in Japan 1 (1961), 28-42.

28. S. Winograd, "On the Number of Multiplications Necessary to Compute Certain Functions", Comm. on Pure and Appl. Math. 23 (1970), 165-179.